Software Cost Estimation:

SLOC-based Models and the Function Points Model

Version 1.1

By
Brad Touesnard

23 February 2004

Table of Contents

1 ABSTRACT 1

2 INTRODUCTION 1

3 SLOC-BASED MODELS 2

3-1 Estimating SLOC 2
3-2 Using SLOC Estimate for Cost Estimation 4

4 FUNCTION POINTS MODEL 4

4-1 Counting Functions and the Calculating Unadjusted Function Points 5
4-2 Calculating the Adjusted Function Points 6
4-3 Interpreting Adjusted Function Points 7

5 CONCLUSIONS 8

6 FURTHER READING 10

1 Abstract

The purpose of this report is to provide an in-depth look at estimating software cost using the Function Points (FP) model as opposed to a more traditional Source Lines of Code (SLOC) -based model. The report will also comment on the advantages and disadvantages of both approaches and their use in industry.

2 Introduction

Estimating software cost is by no means a trivial task and in most cases the larger the software project, the more cumbersome the estimation process. Before the realization of a need for a software cost estimation model, ad hoc models were used for estimating software cost. Today, many small businesses still use ad hoc models while larger businesses tend to embrace a formal model for estimating software cost.

"The most interesting difference between estimation models is between models that use SLOC as the primary input versus models that do not." [1] Source Lines of Code (SLOC) is the oldest metric for estimating project effort and thus is the primary input of older cost estimation models like Putnam's **S**oftware **LI**fecycle **M**anagement (SLIM) from the late 1970's or Boehm's **CO**nstructive **CO**st **MO**del (COCOMO) published in 1981. [1] Using SLOC as input for cost estimation can be problematic simply because estimating the SLOC early in the software development lifecycle can be difficult. Therefore, if the SLOC estimate is inaccurate, the output of the dependant cost estimation model will be inaccurate. Despite these problems, many organizations still use SLOC-based models:

> NASA programs typically measure software size in terms of lines of code. Some authorities recommend other size measures [e.g., function points (see Reference 17)]. However, no other measure is as well understood or as easy to collect as lines of code. [2]

Fortunately for those who do not believe SLOC is an appropriate input for cost estimation, there is an alternative approach that is relatively new. In 1979, IBM's Allan Albrecht published the function points (FP) model which involves "a measure of the amount of function provided by the software system." [3] This model offers several advantages over traditional SLOC-based models and is described in detail in section 4.

Although these models work very well in the environments in which they were developed, often times they do not work well in other situations. Some models have been developed so specifically for their own *native* environment that they can not be generalized for use in other situations. For example, Kemerer could not use the popular PRICE model in his study because it was "developed primarily for use on aerospace applications and was therefore deemed unsuitable for the business applications that would compromise the database."[1] The models that are general enough to be used in a non-native environment must be carefully calibrated in order to yield acceptable results. Kemerer's study found that his results seemed to be skewed in favor of the models that were developed in a similar environment as the environment of the projects that were used in the study.

The remainder of this report will briefly describe the SLOC-based approach to cost estimation and take an in depth look at the FP approach.

3 SLOC-based Models

The estimated SLOC in a proposed software system is used as input to many cost estimation models as described previously in this report. But how are the SLOC accurately estimated in the early stages of the software development lifecycle?

3-1 Estimating SLOC

A SLOC estimate of a software system can be obtained from experience, the size of previous systems, the size of a competitor's system, and breaking down the system into smaller pieces and estimating the SLOC of each piece. [4] Putnam suggests that for each piece, three distinct estimates should be made:

- Smallest possible SLOC – *a*
- Most likely SLOC – *m*
- Largest possible SLOC – *b*

Then the expected SLOC for piece E_i can be estimated by adding the smallest estimate, largest estimate, and four times the most likely estimate and dividing the sum by 6. This calculation is represented by the following formula:

$$E_i = \frac{a + 4m + b}{6}$$

The expected SLOC for the entire software system *E* is simply the sum of the expected SLOC of each piece:

$$E = \sum_{i=1}^{n} E_i$$

where *n* is the total number of pieces. [5]

An estimate of the standard deviation of each of the estimates E_i can be obtained by getting the range in which 99% of the estimated values are likely to occur and dividing by 6:

$$SD_i = \frac{|b - a|}{6}$$

The standard deviation of the expected SLOC for the entire software system *SD* is calculated by taking the square root of the sum of the squares of standard deviations of each estimate SD_i:

$$SD = \sqrt{\sum_{i=1}^{n} SD_i^2}$$

where *n* is the total number of pieces. [5]

3-2 Using SLOC Estimate for Cost Estimation

SLIM and COCOMO are among the many models that make use of a SLOC estimate to estimate software cost in the early lifecycle stages. Unfortunately these models, like most models are highly dependent on the SLOC input and if the SLOC estimate is inaccurate, it will be reflected in the results obtained by the cost estimation model.

Generally to obtain a cost estimate for a software system, three variables are required in addition to the SLOC estimate: alpha, α, the marginal cost per thousand lines of code (KLOC); beta, β, an exponent of the KLOC; and gamma, γ, the additional fixed cost of the project. The cost estimate calculation is represented by the following formula:

$$CostEstimate = \alpha \bullet KLOC^{\beta} + \gamma$$

This is a very basic method for estimating software cost using SLOC, but the details of SLOC-based estimation models are outside the scope of this report. Further reading recommendations are presented in section 6 of this report.

4 Function Points Model

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. 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. [1]

Another important advantage of not making use of SLOC is that the estimate is independent of the language and other implementation variables that are often difficult to take into consideration. To accurately estimate SLOC, the programming language must be considered because some languages are more concise than others. For example, an

estimate of the SLOC for a software project written in Java would undoubtedly differ from an estimate of the same software in Assembly Language. [1]

To properly compare the FP model to SLOC it is important to completely understand how functions are counted, how the final FP count is calculated, and how to interpret the FP count.

4-1 Counting Functions and the Calculating Unadjusted Function Points

Even with the software requirements formally specified, it can be a challenge to get started counting the functions of a software system. To simplify this process, Albrecht provides fives categories of functions to count: *external inputs, external outputs, external inquiries, external interfaces* and *internal files*. [3]

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. [4], [3]

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. [4], [3]

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. [4], [3]

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. [4], [3]

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. [4], [3]

When a function is identified for a given category, the function's complexity must also be rated as low, average, or high as shown in Table 1.

	Low	**Average**	**High**
External Input	__ x 3	__ x 4	__ x 6
External Output	__ x 4	__ x 5	__ x 7
Internal File	__ x 7	__ x 10	__ x 15
External Interface	__ x 5	__ x 7	__ x 10
External Inquiry	__ x 3	__ x 4	__ x 6

Table 1: *Function Count Weighting Factors [6]*

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). [3] This calculation is summarized by the following equation:

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

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

4-2 Calculating the Adjusted Function Points

Although 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 program the system. For example, a software system that requires very high performance would require additional effort to ensure that the software is written as efficiently as possible. [1] Albrecht 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." [6] These characteristics are as follows:

1. Data communications
2. Distributed functions
3. Performance
4. Heavily used configuration

5. Transaction rate
6. Online data entry
7. End user efficiency
8. Online update
9. Complex processing
10. Reusability
11. Installation ease
12. Operational ease
13. Multiple sites
14. 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):

$$VAF = 0.65 + 0.01 \bullet \sum_{i=1}^{14} c_i$$

where c_i is the value of general system characteristic *i*, for $0 <= c_i <= 5$. [6]

Finally, the UFP and VAF values are multiplied to produce the adjusted FP (AFP) count:

$$AFP = UFP \bullet VAF$$

4-3 Interpreting Adjusted Function Points

In practice, the final AFP number of the proposed system is compared against the AFP count and cost of systems that have been measured in the past. The more historical data that can be compared the better the chances of accurately estimating the cost of the proposed software system. [1]

To continuously refine estimation accuracy, it is essential that the actual cost is measured and recorded once a system has been completed. It is this actual cost that enables the evaluation of the initial estimate.

5 Conclusions

Many people believe that counting the functions of a software project is a more logical way to estimate cost than estimating the SLOC and running it through a SLOC-based model. However, some organizations began using SLOC-based models prior to the conception of the FP model and are very comfortable with the SLOC approach. It will be very difficult, if not impossible to convince these organizations that the FP model is superior when their SLOC-based model is producing excellent results for them.

The FP model also has its critics. The process of counting functions in a software system involves some subjective decisions which can differ among individuals within an organization. Some speculate that estimation results for the same software system can vary significantly by individual. According to the author of a leading software engineering textbook, “The function point metric, like LOC, is relatively controversial...Opponents claim that the method requires some 'sleight of hand' in that computation is based on subjective, rather than objective, data...” [6]

Another problem with the FP model that has been identified is the difficulty to automate data collection. Additional efforts to develop automation tools to help in the data collection process are needed.

Kemerer believes that despite its minor deficiencies, the FP model is the software measure that satisfies the need for a robust measurement metric for software cost estimation.

> ...even the current cost is small relative to the large sums spent on software development and maintenance in total, and managers should consider the time spent on FP collection and analysis as an investment in process improvement of their software development capability. [6]

The FP approach seems to present significant advantages over the traditional SLOC approach for estimating software cost. 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 SLOC is a familiar metric or because it takes a little less effort to collect data is probably not good reasoning.

6 Further Reading

All of the sources referenced in this report are highly recommended for further details of their given topic. Sources that were not referenced in this report but are also recommended are as follows:

- International Function Point Users Group (IFPUG) – http://www.ifpug.org
- Function Point Calculator – http://irb.cs.uni-magdeburg.de/sw-eng/us/java/fp/
- A.J. Albrecht, "Measuring Application Development Productivity," *IBM Application Development Symposium*, pp. 83-92, 1979.
- Silvia Abrahão and Oscar Pastor, "Measuring the functional size of web applications," *Int. J. of Web Engineering and Technology*, Vol. 1, No. 1, 2003.

References

1. Chris F. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Communications of the ACM*, Vol. 30, No. 5, May 1987.

2. Mitchell J. Bassman, Frank McGarry, and Rose Pajerski, "Software Measurement Guidebook," *Software Engineering Laboratory Series*, Rev. 1, pp. 21-46, 1995.

3. Unknown, "The Software Measurement Guidebook," *Software Productivity Consortium*, Boston: International Thompson Computer Press, 1995.

4. David Gustafson, "Schaum's Outline of Software Engineering," New York: McGraw-Hill Trade, 2002.

5. L. H. Putnam, "Example of an Early Sizing, Cost and Schedule Estimate for an Application Software System," in *Tutorial, Software Cost Estimation and Life-Cycle Control: Getting Software Numbers, IEEE Computer Society*, New York: Computer Society Press, pp. 102-127, 1980.

6. Chris F. Kemerer, "Reliability of Function Points Measurement. A Field Experiment," *Communications of the ACM*, Vol.36, No.2, pp. 85-97, February 1993.

Chris F. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Communications of the ACM*, Vol. 30, No. 5, May 1987.

Kemerer's study compared models that used SLOC and those that do not. It also addressed the impact of the environment in which a model is developed and whether models can be calibrated for other environments. Kemerer was also interested to determine if the proprietary models were as accurate as the non-proprietary models.

Chris F. Kemerer, "Reliability of Function Points Measurement. A Field Experiment," *Communications of the ACM*, Vol.36, No.2, pp. 85-97, February 1993.

Kemerer's more recent study of the FP model provides great insight into the reliability of FP as a measurement and dismisses many of the common criticisms of the FP model. This study included an excellent description of counting and calculating FP.

Question 1: (10 Marks)

Describe advantages and disadvantages of the Function Points model of cost estimation.

Advantages

- Estimation data available early in software development lifecycle
- Independent of language and other implementation variables
- A non-technical member of the development team can do the estimation

Disadvantages

- Difficult to automate data collection
- Possible subjective counting of function points

Question 2: (10 Marks)

Describe advantages and disadvantages of a SLOC-based model of cost estimation.

Advantages

- Easy to automate data collection
- Easy to understand SLOC input

Disadvantages

- Subjective counting of SLOC
- Estimation experience can have drastic effects on results
- Possible difficulty calibrating for environments other than the environment in which the model was developed

Question 3: (10 Marks)

What are the categories used for counting function points? Give a brief description each category.

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.