

SOFTWARE SIZE ESTIMATION TECHNIQUES

Daniel V. Ferens

Air Force Institute of Technology (AFIT/LSV)
Wright-Patterson AFB, Ohio 45433

ABSTRACT

This paper discusses the issue of software size estimation, a key consideration in estimating software costs. Although software sizing is a relatively new area of interest, numerous techniques have been developed to predict software size. However, the usefulness of various techniques is dependent on the unique situation of the estimator. Furthermore, some techniques are of questionable validity. Nevertheless, the area of software size estimation will continue to receive emphasis and new techniques will continue to emerge.

INTRODUCTION

Software size, usually expressed in source lines of code (SLOC) is the key input to most software cost models. For example, Najberg states, "Although alternatives to lines of code have been proposed over the years, it appears that lines of code will remain the standard for cost estimation purposes" [1]. Furthermore, renowned cost analyst Dr. Barry Boehm states, "The biggest difficulty in using today's algorithmic software cost models is the problem of providing sound size estimates" [2]. Boehm testifies to the importance of software size in that it is the primary input to the Constructive Cost Model (COCOMO) described in his book, Software Engineering Economics [3].

Although the importance of software size has been ascertained, the estimation of software size has often been difficult. It has only been during the past few years that software size estimating techniques have received significant attention. Currently, there are many techniques and models available; however, few models or techniques have been validated or even been shown suitable for even a small range of software projects. The situation is further complicated in that different

models and techniques appear to be applicable for different types of projects during different developmental phases. Therefore, an estimator may be perplexed as to which of the techniques is most appropriate for his or her specific project. The first step to resolving the size estimation dilemma, however, is to understand the various categories of sizing models and techniques available, and the capabilities and limitations within each category.

CATEGORIES OF SOFTWARE SIZING MODELS

As illustrated by Reese and Tamul-
evicz [4], as well as others, there are various schema for categorizing models such as those used for software sizing. The format used here is based on the author's own research and observations of the software sizing field. In this paper, software sizing models are divided into five categories: analogy models, regression models, expert judgment models, models based on function points, and parametric models. For each category, a brief description is given, one or more examples are presented, and the capabilities and limitations of models or techniques in that category are delineated.

1. Analogy Models: Models and techniques in this category estimate size by comparing a program with a similar program or programs of known size. Reifer [5] illustrates a simplistic equation for analogy estimation as follows:

$$S = F \times (\text{Size of Similar Packages}) \quad (1)$$

where "S" is size (usually in SLOC) and "F" is a factor determined by experience or politics.

Since "F" is often difficult to determine, more complex analogy models have been developed which contain extensive historical data bases. Such models are sometimes classified as data base

analogy models [4]. An example of such a model is the Software Size Estimator (SSE) developed by Aerospace Corporation [6]. The SSE model contains a data base of past projects, primarily from space systems. The model user inputs the software type (E.g. spacecraft communications), and a complexity assessment for the software program. The model searches the data base for a similar program or programs, and computes a probable size estimate based on the similar program or programs.

Analogy models have an advantage in that the estimate is based on actual programs for which historical data exists. They are also useful early in a program because minimal input data is required. However, the program being estimated must have a true historical precedent, or the estimate will be misleading. Therefore, analogy models are of limited use programs where the application, language, etc. is new. A further limitation specific to data base analogy models is that they are not useful outside of the scope of the data base from which the model is developed. For example, the SSE model may not be useful for programs other than those with space applications.

2. Regression-Based Models: Models and techniques in this category are similar to data base analogy models in that a historical data base is employed. However, they differ in that regression models do not estimate size by direct comparison to similar programs. Instead, they are developed by performing regression analysis on historical data and deriving size estimating relationships which relate size to one or more input factors.

An example of a model based on linear regression is one developed by Ikatura and Takayanagi [7] from 38 banking programs written in the COBOL language. The "best fit" linear size estimating relationship, according to Ikatura and Takayanagi, is as follows:

$$Y = -810 + 310 X_1 + 1.12 X_2 + 553 X_3 + 5.91 X_5 + 1.62 X_7 + 99.7 X_8 \quad (2)$$

In Equation 2, "Y" is SLOC, and the "X" values represent program characteristics. Specifically, X_1 is the number of input files, X_2 is the number of input items, X_3 is the number of output files, X_5 is the number of transaction type reports, X_7 is the number of horizontal items in tabular reports, and X_8 is the number of calculating processes. Although other "X" factors were considered (E.g. X_4 , the number of output file items), the best correlation was obtained from those factors listed in

Equation 2.

Regression-based models share an advantage with analogy models in that they are based on historical data. An additional advantage is that they can be used for programs which do not have directly analogous programs in the historical data base, since the sizing equations transcend the need for direct analogy. They are also useful early in a program if the input factors are available. However, like analogy models, regression-based models are seldom appropriate for programs outside of the scope of the data base from which they were developed. Equation 2, for example, would probably not be suitable for programs other than those written in COBOL for business applications. Another difficulty with regression-based models is that a valid regression-based equation should have a high correlation coefficient and a low standard estimating error. Unfortunately, for many data bases, this is not always possible. Whetstone [8], for example, could not establish a valid linear regression-based sizing equation for any of the four Air Force data bases studied. An alternative in such cases is to use non-linear regression, but this may result in "forced fitting" of an equation to a data base; a procedure of questionable soundness.

3. Expert Judgment Models: With this technique, one or more experts are consulted for their ideas regarding the size of a program or factors which affect software size. Expert judgment models and techniques may be used alone, or are sometimes used in conjunction with other techniques. For example, expert judgment could be used to determine the inputs for a regression-based model.

One of the most basic expert judgment techniques is sizing using the program evaluation and review technique (PERT) equation. The PERT sizing equation, explained by Boehm [3], Reifer [5], and many others, is as follows:

$$E = \frac{a + 4m + b}{6} \quad (3)$$

where "E" is the expected size (usually in SLOC), "a" is the smallest possible size, "m" is the most likely size, and "b" is the largest possible size. The values for "a", "b", and "m" for Equation 3 are determined by experts.

A more complex expert judgment model is the Software Sizing Model (SSM) developed by Bozoki [9]. SSM requires expert judgment in four areas: PERT sizing estimates, pairwise comparison, program sorting, and program ranking. SSM also

requires two reference programs of known size, plus user inputs for each of the four areas. Another complex expert judgment model, recently developed by Lambert [10], requires prioritized ranking of projects and uses mathematical eigenfunctions to determine ranked project sizes. Lambert's model requires no historical size values; only relative ranking.

One marked advantage of expert judgment models is that they require little or no historical data, enabling their use when such data is not available. They are, therefore, useful for new programs for which historical precedents do not exist. Furthermore, because these models require little "hard" data, they can be useful early in a program. However, expert judgment models are highly dependent on personal opinions, which may be subjective and biased. Also, finding a true "expert" may be difficult; even the assessments of knowledgeable personnel are sometimes mere guesses.

4. Models Based on Function Points:

The function point concept, initially published by Albrecht and Gaffney [11], appears to be a popular approach to software size estimation. Employing a historical data base of 48 COBOL and PL/I programs used in data processing applications, Albrecht discovered that there was a strong correlation between function points and SLOC. An equation derived from the entire data base is as follows:

$$S = 53.2 \times F + 12773 \quad (4)$$

where "S" is SLOC, and "F" is the number of function points. Albrecht and Gaffney [11] have concluded that function points may be superior to SLOC as a predictor of the overall cost or effort for a program.

Function points are computed from examining five program characteristics. According to Behrens [12], an equation for computing function points based on these inputs is as follows:

$$F = 4 \times I + 5 \times O + 4 \times Q + 10 \times M + 7 \times N \quad (5)$$

where "I" is number of inputs, "O" is number of outputs, "Q" is number of inquiries, "M" is number of master files, and "N" is number of interfaces in the program. Equation 5 currently appears to be the "standard" equation for function points computation.

An example of a model which uses function points to compute size is the SPQR/20 model [13]. It uses Equation 5, then adjusts "F" based on user-supplied complexity assessments for the problem, software code, and data. The adjusted

function point count is used to compute a value for SLOC and to compute cost or level of effort required for the program being analyzed.

One limitation of function points is that they are not always applicable outside of the business or data processing environment for which they were developed. For example, the applicability of function points for real-time command and control systems is highly uncertain. Currently, Jones [14] is planning to modify the SPQR/20 model for real-time and system software programs by using feature points. Feature points are a superset of function points which uses an additional input, number of algorithms, and uses different coefficients for the five input factors shown in Equation 5. A currently-available model which also adapts the function point concept to real-time and scientific programs is the ASSET-R model [15]. ASSET-R uses Equation 5 for data processing systems, but replaces the "inquiries" input with "modes" for scientific programs and uses different coefficients for the other four inputs than those coefficients used in Equation 5. For real-time programs, ASSET-R uses the same inputs as those used for scientific programs; and adds two more inputs, stimulus/response relationships and rendezvous, to compute function points. Also, the coefficients differ from those used for either data processing or scientific programs.

The current popularity of function point models undoubtedly stems from the research conducted by Albrecht and others, which demonstrated their propriety for the data processing environment. Current efforts by Jones [14], Reifer [15], and others is tailoring the function point concept to many different types of programs. Proponents of the function point concept claim that the input information is available in the early stages of a program. However, some writers (E.g. [4]) have stated that the input information is sometimes difficult to obtain. Another potential disadvantage to the function point method is that the attempts to apply it to non-data processing applications are relatively recent, and that the applicability to other applications has not yet been firmly established.

5. Parametric Models: These models use input parameters consisting of numerical or descriptive values of selected program attributes. Parametric models are in widespread use for software cost estimating, and are now being used to estimate software size. Many parametric models can be calibrated; suitable values for one or more input parameter can be determined

from historical data.

An example of a parametric model used for software size estimation is the RCA PRICE Sizer (SZ) model [16]. The calibratable PRICE-SZ model contains separate versions for commercial and military applications. For each version, the model requires over 15 inputs which describe characteristics of the program being estimated. For military applications, required inputs include numbers of alphanumeric and graphic displays, input and output streams, control states, and computed tables. For commercial applications, required inputs include numbers of output screens, input and output files, and input fields. Both versions have inputs for requirements growth and technical features. A calibration factor, "SICAL", can be calibrated to a user's organization if sufficient historical data is available. PRICE-SZ computes size in either SLOC or object instructions.

Parametric models have the advantage of considering many different program facets in estimating program size. Also, the calibration capability, when present, allows a user to fine-tune a parametric model to specific applications. The need to quantify input parameters can contribute to objectivity in estimating. Parametric models do have some disadvantages, however. Like function point models, the input data may be difficult to obtain, especially during early stages of program development. Also, since they must be calibrated from historical data, their applicability to new, unique programs may be uncertain.

6. Composite Models: Some models employ a combination of two or more categories of models or techniques discussed above. A current example of a composite model is the Size Planner model developed by Quantitative Software Management [17]. The Size Planner model contains three separate techniques: fuzzy logic, standard components, and function points. The fuzzy logic technique, which embodies selected attributes of expert judgment and data base analogy techniques, combines statistical characteristics of more than one thousand historical programs with the intuitive guesses of the user. It is said to be useful early in a program, where the other two Size Planner techniques may be difficult to use. The standard components technique requires the user to specify twelve independent sizing factors, along with weights for each factor. The technique then compares the user's information with the model's data base, or a data base supplied by the user, to estimate size. Finally, the function points technique

uses Equation 5 plus a complexity value supplied by the user. All three techniques output SLOC for the language specified by the user. The user has a choice of using any of the three techniques in Size Planner individually, or can use a weighted average size computed by the model using all three techniques.

Ideally, composite models such as Size Planner combine the best features of individual techniques while overcoming the limitations by allowing a choice of alternate techniques. However, composite models will still have some limitations, and are not a panacea for all the problems associated with size estimation.

7. Summary of Current Techniques:

Table 1 summarizes the six techniques discussed above, along with one or more examples advantages, and limitations of each technique. As illustrated in the discussions, there are many different models or techniques that can be used for software size estimation. None of the models or techniques presented, however, is inherently superior; the user must determine which available model or technique is most appropriate for his or her application. Furthermore, the rapid evolution of the field of size estimation indicates that additional models will be available in the near future.

CONCLUSIONS AND FUTURE DIRECTIONS

Although the field of software size estimation has matured rapidly, there are still many problems remaining to be solved. The size estimates resulting from any model or technique can be expected to be no better than the quality of the information which is input. For most models, obtaining valid input data is a challenge, especially during the early stages of a program. Furthermore, there is currently a lack of historical sizing data, especially for real-time programs such as those used in national defense. This paucity of data hinders the development of extensive data bases and impedes effective calibration of parametric models. Even when historical data is plentiful, the applicability of the data to new programs can be dubious. Finally, there is the problem of defining size itself. Most models compute size in SLOC, but there is often disagreement as to what type of statements are included in SLOC. For example, most models include executable lines of code in the computation of SLOC, but not all models include other types of code, such as data statements, comments, declaration statements, and job

TABLE 1			
CURRENT SOFTWARE SIZE ESTIMATION TECHNIQUES			
TECHNIQUE	EXAMPLE(S)	ADVANTAGES	LIMITATIONS
ANALOGY	- SSE	- BASED ON HISTORICAL DATA - USABLE EARLY IN A PROGRAM	- TRULY SIMILAR PROGRAM MAY NOT EXIST
REGRESSION	- IKATURA-TAKAYANAGI	- BASED ON HISTORICAL DATA - TRANSCEND NEED FOR DIRECT ANALOGY	- DATA BASE LIMITED - REQUIRE HIGH CORRELATION, LOW STANDARD ERROR
EXPERT JUDGEMENT	- SSM - LAMBERT	- NO HISTORICAL DATA REQUIRED - USEFUL EARLY IN A PROGRAM	- LACK OF OBJECTIVITY - KNOWLEDGE OF "EXPERT"
FUNCTION POINTS	- SPQR/20 - ASSET-R	- WELL-VALIDATED FOR INFORMATION SYSTEMS	- NOT WELL-VALIDATED FOR OTHER SYSTEMS
PARAMETRIC	- PRICE-SZ	- CONSIDER MANY FACTORS - CAN BE CALIBRATED	- SUITABILITY FOR NEW, UNIQUE PROGRAMS - DATA AVAILABILITY
COMPOSITE	- SIZE PLANNER	- FLEXIBILITY - ADVANTAGES OF EACH COMPONENT	- NONE BEYOND LIMITATIONS OF COMPONENTS

control language to the same extent. In other words, there is still not a standard definition of size for most languages.

Some of the current problems associated with software size estimation will be better solved as time progresses. The rapid proliferation of new models will undoubtedly continue, as will efforts to collect additional historical data. With the increased use and emphasis on standard higher-order languages such as Ada, the number of language-peculiar variations in size estimation will diminish. Additionally, a standard definition of what is included in SLOC counts may appear in the near future, at least for some programming languages. However, like the mythical hydra, new challenges can be expected to emerge as old ones are resolved. Increased usage of new software, such as artificial intelligence programs and fourth-generation language programs, will present new challenges, and will prohibit

the field of software size estimation from becoming stagnant.

Perhaps the requirement to estimate size at all may vanish. As Albrecht and Gaffney [11] insinuate, size may not be the best indicator of productivity or effort required. Instead, a measure such as function points may be superior. However, this has not been demonstrated outside of a limited range of programs. For the foreseeable future, therefore, software size estimation will be a necessity, and often a challenge, for software cost estimating and related endeavors.

REFERENCES

1. Najberg, Andrew C., Software Data Base Development, Volume I, Reading, MA, The Analytic Sciences Corporation (Technical Report 4612-S-1): 1984, p. 2-4.

2. Boehm, Barry W., "Software Engineering Economics", Tutorial, Software Management: Third Edition, Washington, DC, IEEE Computer Society Press: 1986, p. 148.
3. Boehm, Barry W., Software Engineering Economics, Englewood Cliffs, NJ, Prentice-Hall: 1981.
4. Reese, Richard M., and Jim Tamulevicz, "A Survey of Software Sizing Methodologies and Tools", ISPA Journal of Parametrics, Vol VII, No. 2, June, 1987, pp. 36-55.
5. Reifer, Donald J., "A Poor Man's Guide to Estimating Software Costs", Tutorial, Software Management: Third Edition, Washington, DC, IEEE Computer Society Press: 1986, pp. 153-163.
6. Singhal, Madhu, "Software Size Estimator", ISPA Journal of Parametrics, Vol VI, No. 4, December, 1986, pp. 38-50.
7. Ikatura, Minoru, and Akio Takayanagi, "A Model For Estimating Program Size and Its Evaluation", Proceedings, Sixth International Conference on Software Engineering, Long Beach, CA, IEEE Computer Society Press: 1982, pp. 104-109.
8. Whetstone, Mark J., Developing Software Size Estimating Relationships Based on Functional Descriptions of the Software (AFIT Thesis GSM/LSY/86S-24), Dayton, OH, Air Force Institute of Technology: 1986.
9. Bozoki, George J., SSM User's Guide, Redwood City, CA, GJB Associates: 1984.
10. Lambert, Joseph M., "A Software Sizing Model", ISPA Journal of Parametrics, Vol VI, No. 4, December, 1986, pp. 75-87.
11. Albrecht, Allan J., and John E. Gaffney, Jr., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Transactions on Software Engineering, Vol SE-9, No. 6, November, 1983, pp. 639-648.
12. Behrens, Charles A., "Measuring the Productivity of Computer Systems Development Activities With Function Points", IEEE Transactions on Software Engineering, Vol SE-9, No. 6, November, 1983, pp. 648-652.
13. SPQR/20 User's Guide, Cambridge, MA, Software Productivity Research: 1986.
14. Jones, Capers, Establishing Enterprise Software Productivity and Quality Measurement Programs (Technical paper), Cambridge, MA, Software Productivity Research: 1986.
15. Reifer, Donald J., ASSET-R: An Overview (RCI-TN-269), Torrance, CA, Reifer Consultants, Inc.: 2 May 1987.
16. PRICE-SZ Reference Manual, Moorestown, NJ, RCA PRICE Systems: 1987.
17. Size Planner (Demonstration Disk), McLean, VA, Quantitative Software Management: 1987.