A Model for Software Sizing

June Verner and Graham Tate
Massey University
New Zealand

Abstract

There is a need for better software sizing models. Two existing models, namely Albrecht's Function Point Analysis and DeMarco's System BANG are examined as candidate models and some of their deficiencies noted. An architecture for a systems sizing model is presented, its major components are identified, and their general functions described.

1.0 The need for better software sizing models.

Notwithstanding the enormous complexity of the problem of software costing, considerable progress has been made in software cost estimation based on a given software size, notably in the COCOMO model [BOE81]. Experience, however, still plays a large part in successful size and cost estimation [BOE82]. Most computer installations are too small to have a separate cost estimation group, have limited access to experts experienced in size estimation, and have inadequate historical records. To make matters worse, some sizing/costing methods are installation-dependent, even to the extent of the basic unit of measurement being defined somewhat differently within the same basic model [DEM82], [ALB79], [COL82], [RUD83]. Many managers are understandably uneasy and defensive about the basis of their estimates. As shown in Figure 1, software sizing is at present the weakest link in the software cost estimation chain. Boehm [BOE81] discusses common reasons for inaccurate sizing. There is a need for software sizing models with levels of reliability similar to those observed for COCOMO [BOE81]. DeMarco [DEM82] defines the qualities of a good metric in this context as being measurable, independent (objective), accountable, precise, consistent and available early enough to be useful.
In the past several years there have been some useful attempts to solve the sizing problem, particularly for commercial data processing systems. In 1979 Albrecht [ALB79] first introduced Function Point Analysis (FPA). He believed that lines of code as a measure was too language-dependent and instead measured the 'size' of a system in function points. In 1982 DeMarco [DEM82] published work on a sizing and costing method that used another unit to measure system size, which he called System BANG. FPA appears to be the more widely used of the two, possibly because it is much simpler to use. It is popular in IBM installations and is increasingly being used for sizing systems written in fourth generation languages.

2.1 Function Point Analysis

The function point measure is said to be based on the user's view of the system and it is claimed that non-DP users can evaluate the measure [RUD83]. The raw function point count is based on the number and complexity of:

- external input types
- external output types
- logical internal file types
- external interface file types
- external inquiry types
This initial count is summed and later modified using 14 system adjustment factors to get a final function point count for the system [ALB84, ALB85]. Since many of these system adjustment factors could be classed as cost drivers rather than size drivers, the final function point count is more a measure of total effort than of size.

FPA was used by Albrecht to compare productivity between projects that were written in different languages and used different technology. He used the average number of lines of code required to develop a function point to show the relative productivities of COBOL, PL/I and DMS/VS [ALB79]. Other writers [RUD83] have used FPA to produce productivity figures for certain 4GLs.

FPA treats the system as if all the inputs, outputs, queries, etc., flow into or out of a single black box process. Thus the raw function point count does not measure the effects of complex processing. FPA tries to overcome this obvious flaw by its use of system adjustment factors. Here the effects of interactive interfaces and complex processing on the system as a whole, together with other cost drivers, are used to modify the raw function point count. We consider these adjustment factors to be the weakest part of the method. They are small additive corrections with equal maximum weight (an unlikely situation in the real world) and with a relatively small total effect, having a possible range of from -35% to +35% adjustment. Given the subjectivity of the adjustment criteria, and the tendency of assessors towards means, the overall adjustment of the raw function points will rarely be more than 15%. These adjustment factors seem rather old-fashioned as they tend to be aimed at batch processing systems.

Albrecht and Gaffney [ALB83] have also used function points, and counts of unique I/O operands, as estimators for delivered source instructions, and effort, in COBOL and PL/I programs.

2.2 DeMarco's System BANG.

DeMarco in [DEM82, DEM84] suggested another measure of system size called System BANG. This measure is calculated from Structured Analysis specifications. The system specifications are developed down to functional primitives (FP). A FP is described as "a trivial piece", "too small to justify further partitioning" [DEM84]. The number of input and output tokens at the boundary of each FP is summed and then used to adjust the value of the FP. DeMarco provides a table of weighted FP increments for this adjustment and states that the values are based on Halstead's volume/vocabulary
relationship [HAL77]. Each FP is then given an empirical Complexity Correction Factor. This factor depends on the class of primitive function. DeMarco defined 16 classes (a beginning set) of primitives although he suggests that the Correction Factor value for some may be less likely to remain invariant than for others. Each weighted FP is multiplied by its appropriate Complexity Correction Factor and the resulting FP values are summed to give total System BANG.

DeMarco found the measure was not installation-independent and suggested that "the complexity weighting factors are, unfortunately, environment dependent" and that two of them, device management and computation, "you correct with your own estimate based on type of computation or type of device" [DEM82]. He also suggests that "you will need to develop your own set of weightings and perhaps some of your own new classes" [DEM82].

DeMarco in [DEM82] suggests a second method to obtaining System BANG. If the system is data-strong i.e. one with a significant database then the BANG can be based on the count of objects in the database. Each object is corrected for the number of relationships at the object boundary and the corrected objects are then summed. For systems that are both data strong and function strong he suggests that the system should be divided, and two sets of BANG metrics be used. The two predictors should not be combined and the project should be treated as if it were actually two projects. He does not believe that there is any satisfactory way of combining Function BANG and Data BANG in the general case but that an individual installation may be able to get a procedure for relative scaling so that the two could be added together.

System BANG has some appeal as it is developed from specifications but the functional primitives are at such a low level that the information from it would not be available early enough to be as useful for feasibility sizing.

System BANG (at least from the published material) appears to be a general approach to a sizing method, that different users can customize in different ways, rather than a cookbook that they can follow. It is still too subjective and the work involved for a DP department to build up its own weightings and classes of primitives would prohibit wider use in its present form. This would result in the measure becoming too installation-dependent. We see the System BANG approach as a very promising one if it can be built into and automated specification tool so that it can be calculated automatically as specifications are developed, e.g. in conjunction with the Structured Systems Analysis and Design Method's MUST set of automated tools being developed.
at present in the UK. Indeed, until it is automated, it is unlikely that it will be used at all widely.

3.0 An approach to software sizing.

The software sizing and costing model shown in Figure 2 provides the framework for our research in software sizing. The great complexity of the whole sizing and costing problem, coupled with the relative success of the COCOMO model in costing from a given size, leads us to concentrate on the estimation of system size in KDSI (thousands of delivered source instructions). Objections have been raised to KDSI as a system size measure [KAF85], [COL82], [BRY83]. However it has the great advantage of providing a measurable output of the sizing model and input to the costing model.

A more fundamental objection can be raised in relation to the sizing and costing framework shown in Figure 2, namely that it assumes that the sizing and costing processes are separable. This assumption has so far been largely vindicated by the success of the COCOMO model. If further research showed that many significant cost drivers were also size drivers, it may have to be questioned.

At this stage, we have restricted our attention to data processing applications, such as those commonly found in commercial processing, implemented in the organic mode. This has the advantage of greatly restricting variability within the domain of interest. It also has the pragmatic advantage for us of ready availability of empirical data. A complication is the increasing variety of languages in which data processing systems are now commonly written, including many so-called fourth generation languages. Often, more than one language is employed in a single system.

We see the software sizing model as consisting of three main components, A, B and C, circled in Figure 2. The inputs to the sizing model are

(i) the system model, which is as far as possible an abstract specification of the system free from implementation considerations or other similar constraints - it describes what the system is to be, e.g. in data flow diagrams, a data model, and associated dictionary entries
A major goal for part A of the sizing model is objectivity. We see no good reason why sizeA should not be computed directly from a computerized system model as a by-product of the system definition process. A critical issue is the unit of measure for sizeA. Candidates include lines of code in a standard language, or a language-independent measure, such as function points, System BANG, or a similar metric. It is our belief that improved language-independent metrics can be developed based on experience using function points and System BANG both of which have deficiencies as noted in 2.1 and 2.2.

Part B of the sizing model has the role of mapping the language-independent sizeA to the language-dependent sizeB, which can be thought of as being the size of a 'neutral' source coding of the system in the chosen language.
several languages are used, or where the mapping is different for clearly definable parts of the system, sizeB is a weighted sum of components. The notion of language level, and work similar to that of Albrecht [ALB83] relating function points to LOC in different languages, is relevant here.

Part C of the sizing model aims to apply appropriate correction factors to sizeB, or parts thereof, to obtain a realistic final size in KDSI. The oft-quoted work of Weinberg and Schulman [WEI74] would suggest that such adjustment factors can, in special cases, be much greater than the + or - 35% limit of FPA. In most systems, large adjustments would, however, only apply to parts of the system, e.g. the user interface.

4. Summary and future work

Existing methods of software sizing have a number of deficiencies, including too great a degree of subjectivity. An approach similar to DeMarco's System BANG may be a candidate for automated software sizing as a by-product of system specification. Work is proceeding in this direction.

This paper presents an architecture for a software sizing model suitable as a front-end to the COCOMO software costing model. It makes explicit the roles of

- the abstract system model(s),
- the language(s) employed, and
- other size drivers

in the computation of KDSI for a software system.

Work is proceeding on the refinement of the sizing model and the collection of data for determining its parameters and eventually validating it.

5. References


ALB83 Allan J. Albrecht and John E. Gaffney, Jr., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science