From performance measurement to project estimating using COSMIC functional sizing

Cigdem Gencel
Charles Symons

Abstract

This paper introduces the role and importance of the measurement of software sizes when measuring the performance of software activities and in estimating effort, etc, for new software projects. We then describe three standard software sizing methods (the Albrecht/IFPUG, MkII FPA and COSMIC methods) that have claimed to be functional size measurement methods and examine the true nature of their size scales. We find that sizes produced by the Albrecht/IFPUG and MkII FPA methods are actually on a scale of 'standard (development) effort'. Only the COSMIC method gives a true measure of functional size and is thus best-suited for performance comparisons across projects using different technologies. We conclude that in principle, different types of scales, one measuring functional size and the other standard effort, may be desirable for the different purposes of performance measurement and for estimating, respectively. It then follows that for estimating purposes it might be attractive to define a COSMIC-enabled estimating process in which weights are assigned to the components of a COSMIC functional size dependent on the technology to be used by the project being estimated and maybe on other factors. The resulting COSMIC 'standard effort' scale for the given technology, etc., might be expected to yield more accurate project effort estimates than using the standard COSMIC functional size scale. Initial promising results for such an approach are also presented.

1. Introduction

The ability to measure a size of a piece of software in a meaningful and reliable way is the foundation-stone for the quantitative management of software activities. The two most important and common uses of software sizes are (a) as a measure of the work-output when determining the performance of software development and enhancement activities and (b) as the starting point for estimating the effort for a software activity. These two primary uses may be illustrated by the following equations.

As regards use (a), one the most important ways of measuring the performance of a software project is to determine the project productivity, defined as follows,

$$Productivity = \frac{Software\ size}{Project\ effort} \quad (1)$$

where 'software size' represents the work-output of the project.

Re-arranging and re-phrasing this equation (1), we get a simplest equation (2) for estimating the effort to develop a new piece of software, i.e. for use (b).

$$Estimated\ new\ project\ effort = \frac{Estimated\ software\ size}{Assumed\ project\ productivity} \quad (2)$$

The size for the new piece of software must be estimated in some way and the 'assumed project productivity' is obtained either by using equation (1) to measure productivity on comparable internal projects, or by using external benchmark productivity data.

This simplest equation (2) for estimating effort can then be refined by taking into account a variety of cost factors to give a more general top-down estimating equation (3).

$$Estimated\ new\ project\ effort = f\left(\frac{Estimated\ software\ size}{Assumed\ project\ productivity}, Cost\ factors\right) \quad (3)$$

The cost factors can be project, product or development organization related. Among the cost factors investigated in a number of studies such as [1][2][3][4][5][6][7]; team size, programming language type, organization type, business area type, application type and development platform have been found to affect the relationship of software product size to project effort at different levels of significance.

All so-called 'top-down' estimating methods such as COCOMO II [8], Putnam's Model/SLIM [9], SoftCost [10], Price-S [11] are derived from this basic equation (3), with varying degrees of sophistication. Whatever the approach, all require as primary input an estimate of the size of the software to be developed.

It would seem obvious that for the above three equations to be used successfully, the same software sizing method must be used for measurements of performance on completed projects or for the external benchmark data, as will be used for estimating the size of a new project.

The selection of a credible size measurement scale is therefore extremely important. An error in estimating the size of the software to be built usually translates directly into an error in the estimate of project effort. All the other cost factors are, of course, important and data on these should be collected to fine-tune the estimation process, but research has shown that size is usually by far the biggest driver of effort.

Measures of software size can be broadly divided into two categories, namely 'functional sizes' and 'physical sizes'.

Functional sizes are supposed to be a measure of the 'Functional User Requirements' (or FUR) of the software. FUR are defined broadly as 'a sub-set of the user requirements that describe what the software shall do, in terms of tasks and services; they exclude technical and quality requirements' [12]. The key point is that functional sizes should be completely independent of all non-functional requirements such as the technology used for the software, project and quality constraints, etc.

Physical sizes measure some characteristic of the actual piece of software such as a count of the number of 'source lines of code' (SLOC) used for the program, a count of the number of modules or object-classes, etc. Clearly, physical sizes depend on the technology used to develop the software.

In principle, functional sizes have two major advantages over physical sizes for our two common uses. First, the technology-independence of functional sizes means that they are ideal for performance measurement, for example when the task is to compare productivity across projects developed using different technologies. Second, estimating a functional size (from the requirements) of a new piece of software can normally be carried out much earlier in the life-cycle of a new project than when a physical size can be realistically estimated to the same accuracy. Functional sizes should therefore be more useful for project estimating purposes, especially early in a new project life-cycle.

In this paper, we elaborate how a single total functional software size figure is obtained using the attribute-definition models of three widely-used Functional Size Measurement (FSM) methods, namely the Albrecht/IFPUG, MkII FPA and COSMIC methods. Then, we

discuss whether these methods really measure the functional size or something else, and the consequences of our conclusions. Finally, we suggest possible ways in which measurements of COSMIC functional sizes could be used for more accurate effort estimation.

2. What do FSM methods actually measure?

Perhaps resulting from the fact that there is no useful definition of software 'functionality', there are many 'Functional Size Measurement' (FSM) methods, all assuming different models of software functionality. Five methods have been published as international standards [13][14][15][16][17] under an ISO policy of 'let the market decide' which method to use. We will describe our three chosen methods in the sequence in which they were developed and only in sufficient detail to illustrate our argument.

FSM methods define two phases for the measurement of software [18]. In the first phase, the FUR of software are extracted from the available artifacts and expressed in the form suitable for measuring a functional size. FUR represent a set of 'Transactions' and 'Data Types' [19]. A Transaction takes Data Type(s) as input, processes them, and gives them as outputs as a result of the processing. FSM methods use Transactions and Data Types as their Base Functional Components (BFC's)[1]. Then each BFC is categorized into BFC Types and the attributes of BFC Types relevant for obtaining the base counts are identified. In the second phase, the functional size of each BFC is calculated by applying a measurement function to the BFC Types and the related attributes. Then the results are aggregated to compute the overall size of the software system.

In the next sub-sections, we discuss the attribute definition models of the Albrecht/IFPUG, MkII FPA and COSMIC methods. FSM methods all define some form of compound measure [20][21] derived by aggregating BFC Types. We therefore discuss what we really measure by these FSM methods.

2.1. The Albrecht/IFPUG methods

The first proposal that one could measure a functional size from its requirements came in a brilliant piece of original thinking from Allan Albrecht of IBM. He named his method 'Function Point Analysis' [22]. The definition of this FSM method has been refined over the last 30 years and is now maintained by the International Function Point Users Group (IFPUG), but the underlying principles of the method have not changed.

The first full account [23] of the model that is still the basis of the IFPUG method proposed that the FUR of a piece of software can be analyzed into three types of transactions, called 'elementary processes' (External Inputs (EI), External Outputs (EO) and External Inquiries (EQ)) and two types of 'Data files' (Internal Logical Files (ILF) and External Interface Files (EIF)). Each of these BFC's is classified as simple average or complex depending on various criteria. To size a piece of software, its BFC's must be identified, counted and assigned sizes which we call 'weights'. The sum of the weighted counts of the BFC's gives the size of the software in units of 'Unadjusted Function Points' (UFP). The weights of an elementary process can range from 3 to 7 UFP and a file from 7 to 15 UFP.

The IFPUG method also still has a further factor, the 'Value Adjustment Factor' (VAF), which attempts to measure the contribution to 'size' of 14 technical and quality requirements. The product of the size in UFP and the VAF gives the size of the software in Function Points (FP).

The important question for this paper is: what basis did Albrecht use to determine the weights of the various BFC types and of the components of the VAF? Some form of

[1] BFC: an elementary unit of FUR defined by and used by an FSM Method for measurement purposes. BFCs are categorized into different types, called BFC Types. [12]

weighting is clearly necessary since the total size measure depends on counting disparate types of BFC. Further, the weights of the BFC types depend in turn on various measures (number of DETs, number of FTRs, number of RETs – see [24]), so you can't simply add the counts of BFC's together across all types.

Albrecht's classic paper [22] focuses on productivity measurement. He describes his size as a 'relative measure of function value delivered to the user that (is) independent of the particular technology or approach used'. He mentions that 'the basis for this method was developed …. from the DP Services estimating experience', but is vague on how the weights were actually derived ('they have given us good results'). However, it is clear from a later paper [24] that the weights were derived from correlations between measures of his 'size and complexity factors' and development effort, for projects developed in the period 1974 - 83. This paper refers to several internal IBM estimating guidelines.

We conclude that the IFPUG method is not, strictly-speaking, a true FSM method. Abran and Robillard also reached a similar conclusion in [25]. The weights were originally derived from measurements on 22 IBM projects; mostly (16) developed using COBOL, all from the domain of business application software. (Note: the VAF is clearly technology-dependent, and has been omitted in the ISO standard for the IFPUG method). The IFPUG method should therefore be correctly termed an 'estimating method', since its size scale was designed to predict effort – see further below.

2.2. The Mark II Function Point Analysis method

The 'MkII FPA' sizing method was developed by Charles Symons who was also working in the domain of business application software [26]. He had used the Albrecht method but found that its size measurements did not adequately cope with very complex transactions that had to navigate through complex file structures. He therefore proposed the MkII FPA method which aims to account for the use of files by transactions, rather than, in part, counting them for their existence within the software boundary as in the IFPUG method [24].

For the MkII FPA method, Symons proposed that the FUR of a piece of software can be analyzed into transactions, called 'logical transactions' (roughly equivalent to Albrecht's elementary processes) which each consist of three components: input, processing and output. To measure the size of the input and output phases of a logical transaction, the number of data element types (DET's) on the input and output are counted, respectively. For the size of the processing phase, the number of logical entity-types that are referenced in the processing is counted (ET's). Note that an important difference from the IFPUG method is that there is no upper limit to the size of an individual logical transaction, thus making it a better measurement scale, it is claimed, for transactions that can vary from extremely simple to extremely complex.

Since we have two attributes defined relevant to measure the sizes of BFCs in this method (DET's and ET's), again they must be weighted before they can be added to obtain the size of a logical transaction. Symons determined the weights to be proportional to the relative average effort to develop one DET and one ET.

The 'relative average effort' was determined using data from 64 projects from 10 different development groups, with the intention that the weights should be truly independent of any particular technology or development approach, etc. In spite of this intention, the MkII FPA method is also not a true FSM method but is, strictly-speaking, an 'estimating method'. (The MkII FPA method also originally included a 'Technical Complexity Adjustment', the equivalent of the IFPUG VAF, but this was discarded as unhelpful.)

2.3. The COSMIC FSM method

In 1998, a group of software metrics experts formed COSMIC, the Common Software Measurement International Consortium. Their aim was to develop a true FSM method, i.e. one that conforms fully to the principles of the ISO standard for FSM [12], and which would be applicable to size software from the domains of business application, real-time and infrastructure software. In view of this cross-domain goal, the COSMIC method was defined using concepts and terminology that are neutral across all the target domains.

The COSMIC method [27] proposed that the FUR of a piece of software can be analyzed into transactions, called 'functional processes' (the equivalent to MkII FPA logical transactions). Each functional process can be analyzed into a set of 'data movements' of which there are four types. Entries and Exits move data into and out of the software respectively. Writes and Reads store and retrieve data to and from persistent storage respectively. Each data movement is presumed to account for the associated data manipulation. (No standard FSM method can properly account for data manipulation, so all are unsuitable for sizing mathematically-rich software).

The COSMIC method BFC type, a data movement, is measured by convention as one COSMIC Function Point (CFP). The smallest possible functional process has size 2 CFP; there is no upper limit to its size. Since the method has only one BFC type, its four sub-types (Entries, Exits, Writes and Reads) are considered to account for the same amount of functionality. And since each has the same unit of size, these can be added without the need for weights. The COSMIC method is a true, maybe the only true, FSM method.

3. Functional Sizes or Estimating Method Sizes?

Although neither the IFPUG nor the MkII FPA methods are true FSM methods, this does not mean they are useless for either performance measurement or for estimating.

As we have seen, both methods in fact measure a form of 'standard effort' with an arbitrary size scale. We do not know how Albrecht established his size scale in absolute terms. The origin seems to be in a weighted questionnaire designed to produce a 'size and complexity factor' for an estimating method; the weights were adjusted so that this factor correlated with effort [24] and the factor was then taken as the size measure. Symons simply adjusted his MkII FPA scale so that it produces sizes similar to Albrecht's in the range 200 – 400 FP. Both then confused matters by claiming their scales measured a functional size and naming their size scales as 'function points'. With hindsight this was a mistake.

In spite of all this, it is perfectly valid, subject to one proviso, to monitor productivity across different projects by comparing actual effort (e.g. in units of person-hours) against a standard effort (on an arbitrary size scale). And, subject to the same proviso, it is valid to use sizes measured on a scale of standard effort as input to a method to estimate actual effort. The proviso is that for the productivity comparison all the projects should really be carried out under the same common conditions (e.g. the same software type, technology, etc) and for estimating the same conditions apply for the project being estimated as for the projects whose performance was used for calibrating the estimating method. That is why it has been perfectly valid, in principle, to use these two methods for the last 20 - 30 years for performance measurement and estimating. The concept of 'standard effort' for a given standard task came from work-study practices pioneered by Frederick Taylor over a century ago [31].

Still, it does place a question mark over whether these measures, whose weights were calibrated on a limited number of projects, over 20 years ago using technology of that vintage, are still best suited for both purposes in today's circumstances.

For example, would one expect a size measurement method calibrated 30 years ago on a set of largely COBOL projects to be ideal for comparing productivity across projects

developed recently using widely different technologies? Would one expect an estimated effort determined by such a method to be the best input for estimating the effort for a software development project that will use modern technology? In fact, some current studies such as [28] discuss how to calibrate IFPUG FP weights to reflect recent software industry trends.

The COSMIC FSM method, being totally independent of technology and measuring a pure functional size, is ideal for our first use (a) of comparing performance of projects using different technologies, etc (assuming it is agreed to be a good measure of software functionality in all other respects). But this does not necessarily make it ideal for our second use (b), to measure a size for input to a new project effort estimate where, perhaps, the size should be related to the technology to be used.

We conclude that ideally, two different measurement scales may be required for our two uses. For use (a) of comparing performance of projects using different technologies, a technology-independent true FSM size scale is needed. For use (b) of project estimating, a standard effort scale that is related to the technology to be used should, in theory, give more accurate results.

It is notable that several well-known estimating methods will accept FP sizes as their primary input, but first convert these to SLOC sizes, which are clearly technology-dependent. These estimating methods have been calibrated using SLOC sizes. But it is also well known that FP/SLOC conversion ratios are fraught with uncertainty [29][30]. Can we do better by having two size scales, one a true standard FSM size scale for performance measurements and the other a locally-calibrated size scale for estimating, where both scales use the same BFC's?

4. Using COSMIC size measurement data for project estimating

Referring to equation 3 above, the factor in the first braces {Estimated software size divided by Assumed project productivity} is actually a measure of 'standard effort' to develop the project according to the sizing method used and given the assumed (standard) productivity; its unit of measurement would typically be 'standard person-hours'. Now the size of a functional process according to the COSMIC method is computed from the following equation.

$$Functional\ size\ (CFP) = N_E + N_X + N_R + N_W \quad (4)$$

The terms refer to the number N of Entries (E), Exits (X), Reads (R) and Writes (W) in the process respectively. The functional size of a piece of software is determined by summing the size of all its functional processes.

However, for estimating purposes, there is no reason to assume that the (standard) effort to analyze, design, develop and test one Entry is the same as that for any of the three other data movement types. And these standard efforts might vary relatively depending on any particular set of 'common conditions' ('C'). Such conditions might include the technology to be used for the software, the 'functional profile' of the software itself (different types of software may have quite different proportions of E's, X's, R's and W's) and maybe other factors. But if, for each condition 'C', we could determine the standard productivity 'S_{CE}', in units of standard person-hours to develop the amount of functionality for one Entry and for each of the three other data movement types, then we can compute the standard effort (in standard person-hours) to develop an amount of functionality (in CFP) for each functional process directly from the following equation;

$$Standard\ effort = (S_{CE} \times N_E + S_{CX} \times N_X + S_{CR} \times N_R + S_{CW} \times N_W) \quad (5)$$

Again, the standard effort required to develop the whole software for the conditions C is obtained by summing the standard effort values for all its functional processes. Our estimating equation (3) can then become:

$$Estimated\ new\ project\ effort = f(Estimated\ standard\ effort, Cost\ factors) \quad (6)$$

Provided the standard effort (in standard person-hours) per CFP for each data movement type are calibrated locally, for the local productivity and for the conditions to be used for the new development, this equation (6) is expected to give more accurate effort estimates than equation (3).

In fact, in our previous studies [32][33][34], we investigated whether effort estimation models based on COSMIC BFC types rather than those based on a single total functional size value would improve estimation reliability by making statistical analyses on the ISBSG dataset [36]. The results of these preliminary studies showed that the improvement in accuracy from using equation (6) rather than (3) is likely to be significant.

In [35], the concept of software 'functional profile' was defined as the relative distribution of its four BFC Types for any particular project. This study investigated whether or not the size-effort relationship was stronger if a project had a functional profile that was close to the average for the sample studied. It was observed that identifying the functional profile of a project and comparing it with the average profile of the sample from which it was taken can help in selecting the best estimation models relevant to its own functional profile. The findings of these studies therefore lend weight to the idea that it is important to take into account a piece of software's functional profile for the best estimating accuracy

In the next paragraphs, we shall give an example approach of how such local calibrations might be done to estimate effort from COSMIC functional size in a software organization. We will also explore how sensitive effort estimation might be, in extreme cases, to using as input counts of COSMIC BFC's weighted by 'standard effort weights' as opposed to using pure COSMIC functional sizes.

5. Calibrating COSMIC Standard Effort weights

In this paper, we explore two calibration methods for COSMIC. These are discussed in the following sections.

5.1. By statistical analysis

First and most obviously, given a set of project data where the counts of BFC Types (data movements of each type, i.e. Entry, Exit, Read and Write) and the actual development effort are available for each project, a statistical correlation of the effort figures versus the corresponding four BFC counts should produce the standard effort weights directly.

Such an analysis depends on having data from a large-enough set of projects that were developed under the same or very similar 'common conditions' (e.g. the same technology, application type of software, etc) as described above. The resulting weights would then be valid for those 'common conditions'. Unfortunately, although plenty of data are available to us, at the moment there are not enough projects with any one set of common conditions for us to use so as to obtain meaningful standard effort weights by statistical analysis. The maximum data points that we could have obtained after forming a subset of similar projects to investigate from the projects data in ISBSG dataset [36] is 11 which is very low to make any meaningful conclusion. Therefore, until we have statistically significant amounts of data to derive the weights, we need another approach.

5.2. By modeling the functional process profile for a particular domain

A second approach is to construct a model, by expert judgment or by using real project data, of the functional processes of the software from a particular domain of interest, and to use the result of the model to calculate the standard effort weights directly. First we describe the general process to construct the model and then we illustrate the process as far as we are able at this stage

A model may be constructed in two Steps, as follows. For each Step, there are two possible approaches, namely either using expert judgment or using real project data.

Step 1 (using expert judgment)

a) Define a set of functional process types that are typical of the domain concerned, and define the relative frequency of occurrence of those functional process types[2]

b) For each functional process type, define the average number of data movements by type (E, X, R and W), i.e. the BFC's

c) Sum the number of data movements over all functional process types for the software, weighted by the frequency of occurrence of the functional processes in which they occur

d) Scale the numbers of data movements by type from the previous step to produce a relative distribution of data movements by type for an average functional process over the domain

Step 1 (using real project data)

e) For a real set of software representative of the domain, measure all the sizes using the COSMIC method.

f) Sum the number of data movements by type over all the software.

g) Scale the numbers of data movements by type from the previous step to produce a relative distribution of data movements by type for an average functional process for the software

Step 2 (using expert judgment to obtain relative standard effort per BFC)

a) Determine, from experience or intelligent guesswork, the relative effort per data movement type (this will almost certainly vary according to the functional process type as determined in Step 1)

b) Weight the output of Step 1d) above, i.e. the relative distribution of data movements by type, by the 'relative effort per data movement type' from the previous step 2a. The result is the standard effort by data movement type for the functional processes of the model

c) Scale the standard effort numbers from the previous step to obtain a relative standard effort by data movement type for the domain

Step 2 (using real project data to obtain calibrated, i.e. actual, standard effort weights per data movement type)

d) Determine, by collecting real project effort data, the distribution of effort over the four data movement types. (This may be difficult to obtain by direct effort measurement on projects; it will probably be necessary to use a 'Delphi' approach, i.e. to ask project team members for their assessment of the proportions of total project effort needed for each data movement type.)

[2] Functional process type: Different types of functional processes might occur in a system depending on the pattern of information processing logic of the software domain. For this study we use the well-known pattern of. functional processes (Create, Read, Update, Delete types) of the business application software domain.

e) Using the total effort for each project, the distribution of effort over the data movement types from the previous step 2d and the distribution of numbers of data movement types from Step 1g, determine the standard effort per data movement type

An Illustrative Model for determining standard effort weights for COSMIC BFC's

We next constructed a model using both expert judgment and real project data, where available, to illustrate the application of the above process to software in the domain of business applications. We carried out Step 1 by both approaches but, lacking real project data, we were only able to calculate relative (as opposed to absolute) standard effort weights at this stage in Step 2.

Step 1 (using expert judgment). We suppose a large portfolio of business applications which has been developed using common conditions. We would expect from the 'CRUD' rule (create, read, update, delete) for the life-cycles of primary entity types the following pattern of functional processes and data movements. (We ignore delete functional process types in this first attempt at a model). There will also be some functional processes to maintain non-primary entity types.

Numbers of functional processes to maintain primary entities

- There will be more enquiry ('read') functional processes than 'inputs' (creates and updates), because for all data that is entered it must be possible to enquire about it and, in addition, there are all the enquiries on derived and aggregated data (data that is not stored)
- There will be more updates than creates because many primary entities have several stages to their life-cycle

Numbers of data movements of these functional processes

- For every 'input' functional process there will be one or more Entries. For every Entry in the functional process there will be:
    - the same number of Writes
    - some Reads (usually more than the number of Entries) for validation of entered data
    - an Exit for error messages
- For every 'output' functional process there will be
    - One Entry, one or more Reads and one or more Exits, probably more Exits than Reads, especially if an error message is included

Functional processes to maintain secondary entities

- A few simple functional processes of simple data movements

Now we can build (guess) a model of an 'average functional portfolio' to determine the relative number of functional processes (FP) and data movements (DM) (see Table 1).

Step 1 (using real project data). In order to show an example for the approach using real project data, we obtained data from the ISBSG database on 22 business application projects measured using the COSMIC method and added data from 4 more business application projects we measured.

We obtained the relative number of data movements based on the average number of E, X, R and W and normalized the counts with respect to the number of E. In Table 2, we provide the results from both approaches.

Table 1. Average functional portfolio model

Type of FP	Relative # FP's*	#E's	#W's	#R's	#X's
FP's to maintain primary entities					
Create	1	1.5	1.5	2	1
Update	1.5	1.5	1.5	2	1
Read	2	1	0	2	3
FP's to maintain secondary entities					
Create	0.1	1	1		1
Read	0.1	1		1	1
Delete	0.1	1	1		1
#DM's weighted by relative #FP's		6.05	3.95	9.1	8.8
Relative # DM's (normalized)		1.00	0.65	1.50	1.45

* (# = number of)

Table 2. The relative number of data movement types by the two approaches

Data Source	Relative number of data movements			
	#E's	#W's	#R's	#X's
Expert judgment model	1.00	0.65	1.50	1.45
26 business application projects	1.00	0.53	1.22	1.37

Considering that the 'expert judgment model' is an unrefined first attempt and that the 26 projects are a far-from-homogeneous set, the agreement between these two sets of figures is encouraging. The reader is welcome to substitute his own expert judgment model.

Step 2 (expert judgment) We assume the following effort per data movement (see Table 3).

- The effort for a Read is the same, on average, as that for a Write
- Every Entry of an Input functional process requires more effort than every Exit of an Output functional process, since the former involves validation as well as formatting.

We assume the following for the relative effort needed to develop each DM type.

Maintain primary entities

- 'V Hi' for E's in create and update FP's
- 'Hi' for X's in Read FP's, except for all error messages which are 'Lo'
- 'Lo' for E's in Read FP's
- 'Med' for all other DM types

Maintain secondary entities

- 'Lo' for all DM's

where we judged that the effort proportions of V Hi, to Hi, to Med to Lo are: 1.33 / 1.00 / 0.5 / 0.1, respectively. We initially classified the relative efforts as 'V Hi', 'Hi', 'Med' and 'Lo' for ease of understanding, but this is an ordinal scale. We therefore devised a ratio scale based on the relative proportions of effort.

Applying these proportions to the above table, where 'Eff' = Effort

Table 3. The relative standard effort weights for the DMs

Type of FP	Relative # FP's	Eff. E's	Eff. W's	Eff. R's	Eff. X's
FP's to maintain primary entities					
Create	1	2.00	0.75	1.00	0.1
Update	1.5	2.00	0.75	1.00	0.1
Read	2	0.10	0	1.00	2.1
FP's to maintain secondary entities					
Create	0.1	0.1	0.1		0.1
Read	0.1	0.1		0.1	0.1
Delete	0.1	0.1	0.1		0.1
Total standard effort per DM type weighted by #FP's		5.23	1.90	4.51	6.28
Relative # DM's (from table above)		1.00	0.65	1.50	1.45
Relative av effort per DM type		5.23	2.90	3.00	4.32
Relative av. effort per DM type (normalized)		1.74	0.97	1.00	1.44

For Step 2, using this expert judgment model for the (rounded) relative standard effort weights for the four types of data movement, we conclude that a single Entry requires 1.7 times as much effort to develop, and that an Exit requires roughly 1.4 times as much effort to develop, as does a single Read or Write.

We then tested how sensitive an estimation method would be to using equation 3 above (i.e. using a pure COSMIC functional size as input) versus using equation (6), (i.e. using a standard effort figure as input). We did this by comparing the effort estimate for two pieces of software, one dominated by 'Create' and 'Update' functional processes (see Table 4) versus a piece of software dominated by 'Read' functional processes (see Table 5). This is a fairly extreme assumption.

Table 4. *Case 1- A software project dominated by create and update transactions to maintain primary entities (ignoring table maintenance)*

Parameter	E's	W's	R's	X's	Total
# of C or U DM's on primary entities	1.5	1.5	2	1	
Functional size (FS) per DM type	1	1	1	1	
Relative av. effort per DM type using FS as the measure of effort	1.5	1.5	2	1	6
Relative av. effort per DM type from above	2.00	0.75	1.0	0.10	3.85

Table 5. *Case 2 - A software project dominated by read transactions on primary entities*

Parameter	E's	W's	R's	X's	Total
# R DM's on primary entities	1	0	2	3	
Functional size (FS) per DM type	1	1	1	1	
Relative av. effort per DM type using FS as the measure of effort	1.5	1.5	2	1	6
Relative av. effort per DM type from above	0.10	0.00	1.00	2.10	3.20

The relative error in estimated effort from using a functional size versus using a standard effort measure for the software systems in these two tables is therefore 3.85 / 3.2 = 1.20, i.e. the relative error could be 20 percent.

We must emphasize that these results are very preliminary, for a particular software domain. We would not expect to obtain the same results for e.g. telecommunications or process control software, which might have quite different distributions of functional process types and data movement types than we have assumed in our illustrative model.

6. Conclusions

We have explored three main methods of sizing software from their requirements and have concluded that two methods (Albrecht/IFPUG and MkII FPA) actually produce sizes on a scale of relative standard effort. Whilst these size scales can be used for performance measurement and for estimating, they are not ideal for either use given they were calibrated using data from relatively small sets of projects from one particular software domain, some 20 – 30 years ago.

The COSMIC method, on the other hand, measures a pure functional size and is thus ideal for carrying out performance measurement comparisons of projects developed using different technology, etc. Functional sizes measured using the COSMIC method can also be used as input to estimating methods. However, our first observation above that it seems 'obvious' that the same size measurement scale should be used for both performance measurement and for estimating purposes turns out to be sub-optimal. We now conclude that the use of pure COSMIC functional sizes as input to an estimating method could be improved by first weighting the BFC counts-per-BFC-type for the piece of software being estimated by a different 'standard effort' per BFC type.

The standard effort figures would be expected to be quite different for different software domains and even different when using different technologies within those domains. Standard effort figures should therefore be calibrated separately (and ideally locally) for each set of 'common conditions', i.e. for each domain and technology combination that is important to the measurer.

This first exploratory study indicates that using weighted rather than un-weighted COSMIC BFC's could increase the relative accuracy of estimated effort (other factors being equal) across different types of software within the same domain and using the same technology by a range of up to 20% in the extreme, which is very significant. The size of the improvement in estimating accuracy would be expected to vary relatively for estimates for software from different domains and/or when using different technologies within the same domain. We are therefore encouraged to continue this line of research.

References

[1] Angelis L., Stamelos, I. Morisio, M., "Building a Cost Estimation Model Based on Categorical Data", 7th IEEE Int. Software Metrics Symposium (METRICS 2001), London, April 2001.

[2] Forselius, P., "Benchmarking Software-Development Productivity". IEEE Software, Vol. 17, No. 1, Jan./ Feb. 2000, pp.80-88.

[3] Lokan, C., Wright, T., Hill, P.R., Stringer, M., "Organizational Benchmarking Using the ISBSG Data Repository". IEEE Software, Vol. 18, No. 5, Sept./Oct. 2001, pp.26-32.

[4] Maxwell, K.D., "Collecting Data for Comparability: Benchmarking Software Development Productivity", IEEE Software, Vol. 18, No. 5, Sept./Oct. 2001,pp. 22-25.

[5] Morasca, S., Russo, G., "An Empirical Study of Software Productivity", In Proc. of the 25th Intern. Computer Software and Applications Conf. on Invigorating Software Development 2001, pp. 317-322.
[6] Premraj, R., Shepperd, M.J., Kitchenham, B., Forselius, P., "An Empirical Analysis of Software Productivity over Time", 11th IEEE International Symposium on Software Metrics (Metrics 2005), IEEE Computer Society, 2005.
[7] Card, D.N., "The Challenge of Productivity Measurement", Proc. Of the Pacific Northwest Software Quality Conference, 2006.
[8] Boehm, B.W., Horowitz, E., Madachy, R., Reifer, D., Bradford K.C., Steece, B., Brown, A.W., Chulani, S., Abts, C., "Software Cost Estimation with COCOMO II", Prentice Hall, New Jersey, 2000.
[9] Putnam, L.H., "A general empirical solution to the macro software sizing and estimating problem", IEEE Transactions on Software Engineering, July 1978, pp. 345-361.
[10] Tausworthe, R., "Deep Space Network Software Cost Estimation Model", Jet Propulsion Laboratory Publication 81-7, 1981.
[11] Park, R. E., "PRICE S: The calculation within and why", Proceedings of ISPA Tenth Annual Conference, Brighton, England, July 1988.
[12] ISO/IEC 14143-1:2007, "Software and Systems Engineering - Software measurement - Functional size measurement - Definition of concepts", The International Organization for Standardization, 2007.
[13] ISO/IEC 19761:2003, Software Engineering – COSMIC-FFP: A Functional Size Measurement Method, International Organization for Standardization, 2003.
[14] ISO/IEC 20926:2003, Software Engineering-IFPUG 4.1 Unadjusted Functional Size Measurement Method - Counting Practices Manual, International Organization for Standardization, 2003.
[15] ISO/IEC 20968:2002, Software Engineering - Mk II Function Point Analysis Counting Practices Manual, International Organization for Standardization, 2002.
[16] ISO/IEC 24570:2005, Software Engineering - NESMA functional size measurement method version 2.1 – Definitions and counting guidelines for the application of Function Point Analysis, International Organization for Standardization, 2005.
[17] ISO/IEC 29881:2008, Software Engineering - FiSMA functional size measurement method version 1.1, International Organization for Standardization, 2008.
[18] Demirors, O., and Gencel, C. "Conceptual Association of Functional Size Measurement Methods", scheduled for publication in IEEE Software, 2009.
[19] ISO/IEC TR 14143-5: Information Technology- Software Measurement - Functional Size Measurement - Part 5: Determination of Functional Domains for Use with Functional Size Measurement, 2004.
[20] Habra, N., Abran, A., Lopez, M., Sellami, A., "A framework for the design and verification of software measurement methods", Journal of Systems and Software, Vol.81 , Issue 5, May 2008, pp. 633-648.
[21] Kitchenham, B., Pfleeger, S.L., Fenton, N., "Towards a Framework for Software Measurement Validation", IEEE Transactions on Software Engineering, Vol.21, No.12, Dec. 1995, pp.929-944.
[22] Albrecht, A.J., "Measuring Application Development Productivity", IBM Application Development Symposium, Monterey, California, October 1979.
[23] Albrecht, A.J. and Gaffney, J.E., "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.
[24] Albrecht, A, J., "Where Function Points (and Weights) Came From", (hand-written slides from a presentation dated 19th February 1986).
[25] Abran, A., and Robillard, P.N., "Function Points: A Study of Their Measurement Processes and Scale Transformations", Journal of Systems and Software, Vol. 25, 1994, pp.171-184.
[26] Symons, C.R., "Function Point Analysis: Difficulties and Improvements", IEEE Transactions on Software Engineering, Vol 14, No. 1, January 1988, pp. 2-11.

[27] COSMIC, "The COSMIC Functional Size Measurement Method", version 3.0, Measurement Manual', obtainable from www.gelog.etsmtl.ca/cosmic-ffp, 2007.
[28] Xia, W., Capretz, L.F, Ho, D., Ahmed, F., "A new calibration for Function Point complexity weights", Information and Software Technology, Vol. 50, Issue 7-8, June 2008, pp. 670-683.
[29] Santillo, L., "Error Propagation in Software Measurement and Estimation", in IWSM/Metrikon 2006 conference proceedings, Potsdam, Berlin, Germany, 2-3 November 2006.
[30] Dekkers, C. and Gunter, I., "Using Backfiring to Accurately Size Software: More Wishful Thinking Than Science?", IT Metrics Strategies, Vol. VI, No.11, 2000, 1-8.
[31] Taylor, F.W., "Principles of Scientific Management", Harper & Bros., 1911.
[32] Gencel, C., and Buglione, L., "Do Different Functionality Types Affect the Relationship between Software Functional Size and Effort?, MENSURA 2007, LNCS 4895, Springer-Verlag Berlin Heidelberg 2008, pp. 72–85.
[33] Buglione, L., and Gencel, C., "Impact of Base Functional Component Types on Software Functional Size based Effort Estimation", 9th Intern. Conf. on Product Focused Software Process Improvement (Profes 2008), 23-25 June, Rome, Italy, LNCS 5089, Springer-Verlag Berlin Heidelberg 2008, pp. 75–89.
[34] Gencel, C., "How to Use COSMIC Functional Size in Effort Estimation Models?" Selected papers of IWSM / MetriKon / Mensura 2008, LNCS 5338, 2008, pp. 205–216.
[35] Abran, A., Panteliuc, A., "Estimation Models Based on Functional Profiles. III Taller Internacional de Calidad en Technologias de Information et de Communications, Cuba, February 15-16, 2007.
[36] ISBSG Dataset 10, http:// www.isbsg.org, 2007.