Using "Backfiring" to Accurately Size Software -- More Wishful Thinking than Science?

By Carol Dekkers, CFPS, Quality Plus Technologies, Inc. and
Ian Gunter, Numerical Science

November, 2000

Abstract: *Functional Size Measurement is a fairly recent concept to be embraced by the information technology industry. But increasingly the method of Function Points Analysis (FPs), as maintained by the International Function Point Users Group (IFPUG), is establishing a position as the gold standard of software measurement. At the same time, in an industry predominant with engineers, computer scientists and math majors, it is easy to understand why physical measures of software size, such as Source Lines of Code (SLOC) continue in common usage. To make the transition between SLOC and FP easier, a method called "Backfiring" was developed to calculate FP by taking the SLOC count and multiplying it by a static factor based on the dominant software programming language. This article presents:*

The basis for the two measures: Function Points and SLOC, highlighting their differences and distinct advantages; and

Analysis highlighting why "backfiring" can lead to gross inaccuracies when sizing software.

Introduction: The Requirement for Different Software Size Measures

Functional Size Measurement, the act of measuring the size of software based on its logical user functions, is a fairly recent concept to be embraced by the information technology industry. Although it was first introduced over 20 years ago in 1979, only in the past 5 years has the method stabilized to such an extent that over 50 commercial estimating tools and multiple industry databases now include function points as one of their critical input parameters. Additionally, Scientific American recently featured (Dec. 1998) a full length article on software sizing that prominently profiled the power of function point based metrics. Today, the Function Point Analysis method as maintained by the International Function Point Users Group (IFPUG) is making inroads for software sizing within the U.S. Department of Defense and the Software Engineering Institute's Capability Maturity Model Integration (CMMI) projects. With the inclusion of function point based metrics in major outsourcing contracts, Function Points are becoming the acknowledged gold standard of software measurement co-existing alongside of traditional physical measures of size such as source lines of code (SLOC).

Why is software size important to the Information Technology industry? Size underpins many key project decisions from work effort and cost estimating to scheduling. As one of the key input measures for predicting project costs, it is vitally important that the anticipated project size be reliable and accurate. However, in our haste to arrive at "quick and dirty" estimates, the importance of an accurate size measure often goes unrecognized and may even be overlooked by estimators demanding quick answers. This has created interest in techniques for generating derived measures of functional software size from other measures using a shortcut approach. In particular, to make the transition between SLOC and FP easier, a method called "Backfiring" was developed that *derives FP* by simply multiplying SLOC by a static factor based on the dominant software development language. The promised benefits of such backfiring techniques are speed and ease of derivation over the manual process of function point counting. But -- using short cuts without a thorough understanding of the limitations often leads to inferior results. In addition, when a "conversion" is attempted between two measures such as SLOC and Function Points, the results can appear to be sound, yet must be challenged in terms of their accuracy and applicability. This is especially true when backfired function points are used as the basis of corporate decision-making.

What are Source Lines of Code and Function Points?

To understand the problem and the associated issues involved with deriving one software size measure (FP) from another (SLOC), it is important to understand the differences between the measures themselves. Source Lines of Code and Function Points measure two distinct and different dimensions of software size; SLOC measures the physical, implemented size of software while FP measure the functional size based on user software functions. As such, the two measures are not *directly* interchangeable – but each has its own special features, advantages and disadvantages. Does this mean that one cannot approximate or derive one measure from another? This subject is the essence of this article, particularly in relation to the backfiring of SLOC into FP.

For ease in understanding, a construction analogy can be useful: Function Points represent the functional area of the logical software requirements in a way similar to how square feet represent the size of a building's floor plan. SLOC, on the other hand, represents the physical size of software similar to how the number of sheets of drywall or feet of copper water pipes are dimensions describing a building's physical size. Some size dimensions can be translated into other dimensions, such as square feet into feet of copper pipe, however, problems can occur if care is not taken to understand what the translated measures mean.

A further examination of SLOC and FP follows which will lead into our discussion of backfiring.

Key features of SLOC

Source-lines-of-code (SLOC) are a measure of the physical size of software. To measure SLOC one counts up the number of non-comment, self-contained lines-of-code contained in the software regardless of whether it is batch or on-line code. Often the total SLOC for a piece of software is subdivided by programming language for ease in "backfiring" the figure into function points. Compiled code and other variations of the source code are not usually counted, but rules surrounding SLOC counting do not usually address whether job control language, hardware specific and other variations of batch submission code is to be counted. In short, the major characteristics of SLOC include that they:

- Are a physical size measure of software based on a count of its source code implementation
- Are simple and easily understood measures in common usage
- Provide a "builder" perspective on the software size based on how programmers view software (similar to how a plumber would view the size of a house based on the number of feet of pipe)
- Are easy, inexpensive to count and automatable, but there are no industry wide standard counting rules in place
- Can be estimated at the coding phase or later, however, actual SLOC figures are not available until later in the development lifecycle
- Are meaningful only for comparisons of software developed in the same language and using similar coding conventions
- Are the primary sizing inputs for many conventional software costing models including COCOMO, COCOMOII, SLIM™, Price/S™, Estimate Professional™, KnowledgePlan™, etc.
- Are appropriate for software sizing when a physical measure is needed, or when large amounts of SLOC data are available, or when measuring maintenance of applications that use the same or similar programming languages
- Are dependent on the programming language and physical implementation
- Vary with the skill and programming style of the individual programmers
- Treat all lines of code equally (i.e., given the same weight)
- Physical size of code "feels" as if it will directly correlate to work effort

Key features of Function Points (FP)

The term "Function Points" refers to the unit of measure that is used to quantify the logical, functional size of software, independent of its development or implementation technology. The Function Point measure is backed by a rigorous method of counting rules maintained by the not-for-profit International Function Point Users Group (IFPUG) which produces the Counting Practices Manual (CPM) currently in release 4.1.[1] In short, function points:

- Measure the functional size of the software, from a user-perspective (what is going to be built)
- Are conceptually less easy to understand than SLOC, especially from a programmer's or developer's point of view
- Provide a software "customer" or "user" perspective on the functional software size
- Are relatively expensive to measure compared to SLOC. To count FP, one needs a skilled counter trained in the FP counting rules.
- Are subject to standard, well-defined counting rules as published by the International FP Users Group (IFPUG)
- Can be *estimated* early in the project/development lifecycle, and *counted* once the requirements are articulated
- Are a primary input for many software costing and work effort estimating models
- Are independent of the programming language and physical implementation – the common currency of software measurement
- Are a more sophisticated measure giving different weight to different types of logical user functions

The key thing to appreciate is that the two measures represent different dimensional attributes of the software, are used for different purposes, and are not interchangeable – rather like height and weight. When we talk about FPs and SLOC, we're not talking about feet and metres, we're talking about square feet and pallets of building drywall.

Why Derive one Software Sizing Measure from Another?

There are many situations where only one directly counted measure of software size, typically the Source Lines of Code (SLOC), may be available. The most common reasons for this include:

- SLOC counts are readily available using automated SLOC counting software
- Certified FP Specialists (CFPS) are not readily available and/or are too expensive
- Lack of time and budget to hand count the FP

[1] The IFPUG Counting Practices Manual, CPM 4.1 (1999) can be obtained directly from IFPUG at www.ifpug.org or by calling the administrative office at (609) 799-4900.

- Lack of understanding about SLOC counts
- No perceived, compelling reason to handcount FP
- Little or no user or requirements documentation from which to generate FP counts
- Out-of-date user manuals or lack of knowledgeable resources from which to glean functional requirements

However, the existence of SLOC alone is inadequate as more and more software cost and work effort estimation models rely on Function Point measures of software size. In such circumstances there may be compelling reasons for wanting to derive Function Points from SLOC. To make such a transition between SLOC and FP easier, a method called "Backfiring" was developed that "calculates" Function Points by multiplying SLOC by a static factor based on the dominant software development language.

For instance, backfiring is often used when only the SLOC figures are available, but FP analysis is not possible or practical. Because the FP measure is independent of the language used to implement the software, Function Points allow comparisons across diverse systems, and offers a wider currency of application than SLOC. Backfiring promises a quick and inexpensive means of deriving a FP figure, relatively effortlessly.

Conversely, the backfiring technique is also used for those times when SLOC figures are needed at the start of a project for input to those cost estimating models that require lines-of-code counts. At the time of estimating, only SLOC estimates are available, and these are often based on questionable, or even unstated, assumptions. The attraction of a SLOC figure derived from an available FP figure is that its derivation appears more methodical and traceable, thus providing figures more credible than other estimates.

The Mathematical Principles of the Backfiring Method

Establishing a mathematical form of the relationship between the FPs measure of software size and the SLOC measure for software implemented in the *same* language is not difficult. The SPR website cites that the constants in their "Programming Languages Table" must be based on a sample size of at least 10 similar language projects, and that the data is continually being refreshed.[2] The major concept behind the backfiring model is the assumption that there is a directly proportional (linear) relationship that can be established for a given programming language. Such a linear relationship is given by

$FP = k * SLOC$ where SLOC is a count of the logical lines of code in the subject software, and k is a constant based on the programming language.

[2] SPR's website is www.spr.com. Refer to their Resources page for further details about their programming languages translation table.

With a set of several data pairs of corresponding FP and SLOC measures for software developed in the same language, the value of the constant translation factor, k, can be calculated via a least squares best fit approach. Once the value of k is established, the model can be applied to derive an estimate of software size measured in FPs from a known SLOC figure, i.e. to “backfire”. However, the calculation of k is not sufficient. No model for use in estimating is complete without some measure of the goodness of fit of the model to the data, and in the case of backfiring confidence limits for the value of k should always be stated.

Similarly it is possible to calculate a corresponding model to derive SLOC from a known FPs figure, i.e. in the form

$$SLOC = k'*FP.$$

Capers Jones, Chairman and Founder of Software Productivity Research (SPR, Inc.) is the father of the backfiring method and he has produced a programming languages table that contains the translation factor values as determined by SPR, Inc. research over the years. In the preamble about how to use and the cautions associated with the use of backfiring, Mr. Jones discusses how the SLOC count is intended to be the count of the *logical* lines of code, (i.e., non-commented, non-compiled program code), not the physical lines.

The Inherent Uncertainties with the Backfiring Method

The uncertainties attached to any model representing the relationship between FPs and SLOC must be understood if the model is to be applied effectively and to meet the intention for which the model was designed. Ultimately, the resultant size (in FP if using the constant k to derive FP from SLOC, or in SLOC if using the inverse constant, k', to derive SLOC given FP) depends on the data which has been used to calculate the value of the appropriate translation factor. The questions that arise in relation to these constant translation factors surround the following issues:

1. What software does the data relate to? (what type and size of application, what development environment) – e.g. is the software to which the backfiring method is being applied of the same size/type, and being developed in a similar environment? If the answer is 'no', then the constant k (or k') may be flawed.
2. Errors in the data – e.g. even with counting rules, +/- approximately 10% for FPs (accuracy according to Chris Kemerer of MIT in one of his FPA studies circa 1993), and SLOC figures subject to no counting rules – is all of the data used to derive the k constant consistent?

3. The goodness of fit of the model to the data - What are the confidence limits for outputs from the model? Is there any flexibility to adjust the constant k for changes in project conditions besides purely programming language level?

Of course, the relationships and the usefulness of the backfiring model all depends on the homogeneity of the data, but in practice, the correlation between software size measured in SLOC versus FPs is far from perfect. So, given that the SLOC to FP translation is flawed on an individual project by project basis, are there, perhaps, some usage situations where backfiring can be used for success. Pallets of drywall translated into square feet of building works well only with a large sample size and a translation constant derived from many, many projects with similar attributes. Otherwise, it appears that the results will only ever be as good as the underlying data on which the translation factor was based.

Acquiring a Suitable Model / Language Translation Factor

There are two options for acquiring the translation factor – calculate your own or use a reliable factor that someone else has derived. Conveniently, tables of language translation factors are available (e.g. Capers Jones). Alternatively, an organisation may have sufficient software size data of its own to consider calculating its own value for a translation factor. Both approaches have their limitations and risks.

Using a published table of translation factor is attractive – all of the statistical analysis has been done – hasn't it? Well, maybe. Usually there are important omissions from published tables. The trouble is they don't tell you where the translation factor has come from – what data was used to calculate it, and you don't know the confidence limits which apply to it.

Equally disquieting is the variation in translation factors in language tables from different sources. What's more, the use of a translation factor from a published language table has no guarantee of success within any particular organization.

If a sufficient supply of relevant corresponding pairs of SLOC and FP measures is available, then it may be better for an organisation to use its own data to calculate an appropriate translation factor. This has the advantage that the relevance of the model is clear, and the goodness of fit of the model can be known. Another benefit of using local data is that it can be used very effectively to test the validity of translation factors from published language tables.

Software Project Attributes Ignored by Backfiring

For all its mathematical rigour, backfiring completely disregards the nature of software projects. The method takes no account of the process which translates the functional requirement into the software implementation, other than the language used for coding the software. Backfiring relies on a linear model of SLOC to FP (or vice versa) where the constant is based solely on programming language. The relationship between SLOC (a physical measure) and FP (a logical, functional measure) is much more complex than backfiring credits to it. Consequently, a single dimensional model such as backfiring can be grossly inaccurate.

Also, there is a wide variation in implementation styles that can affect the SLOC to FP ratio significantly. For example, backfiring will make a product design that is implemented via verbose coding practices appear to be a functionally rich product. What's more, it will conceal instances where the application's product design extends beyond its functional requirements. The larger the number of SLOC program statements, the larger the resultant FP count -- and this may not actually be the situation. While it is easily understood that the larger the project, the larger the source code usually is, the relationship between FP and SLOC is not as simple as it first appears.

A frequent complication when choosing to use the backfiring technique is that many applications are developed using more than one language. Unless there is a well-understood boundary between the separate language elements in the software implementation, the use of backfiring in these situations must involve further approximations.

The simple truth about FPs "backfired" from SLOC is that the technique produces a FPs figure in name only, reflecting a particular programming language only, and representing the software implementation rather than the functional requirement.

Misconceptions and Dangers

In the preceding paragraphs, we have explored the fact that the relationship between SLOC and FPs is less than perfect, and when disparate sets of data are involved how much more inaccurate the figures can actually be. However, despite this, some factions of our software estimating industry tend to talk up the validity of backfiring.

Most software cost estimating tools require SLOC figures as a primary input. But many of these tools have been adapted to accept a FP figure and derive the necessary SLOC figure from it. While the vendors of cost estimating tools are keen to promote this feature as a selling point for their application, the accuracy of the approach is usually played down.

In environments where there are immature metrics programs, the inclination to use backfiring to derive FP figures is strong. And often such backfired figures are used for comparisons with industry averages, as in benchmarking. However the results are likely to be misleading and unfair. In this situation it is the vendors of benchmarking services who are most keen to play down the risks in backfiring.
The use of backfired FP figures in productivity calculations is contentious, especially at the client/vendor interface. For example, in benchmarking an outsourced development and support service the use of backfired figures is likely to be the basis of many arguments between client and vendor.

One question remains: If the backfiring method to derive FP from SLOC (or vice versa) is so flawed, why are major, multimillion dollar outsourcing contracts using it to establish the size of their outsourced portfolio? The answer lies in the fact that some data (even imperfect) is better than no data, and the fact that there are some portfolio wide situations where backfiring can be used to degrees of success. Such instances include corporate wide measurement initiatives where the overall *portfolio SLOC* counts are available, but the organizations involved do not have the time, energy or budget to properly fund a portfolio function point sizing effort. However, in the same way that the number of pallets of drywall used to build an entire village can likely be related to the square foot size of the village with some degree of accuracy because there may be many "average" homes where the relationship is fairly static. As such, a linear relationship between the overall village size and number of pallets of drywall can work -- given a large sample size and a relatively homogeneous environment. This is much the same situation when a portfolio is brought forward to be sized using the backfiring technique. BUT… in the same way that an aircraft hangar introduced into a village will skew the relationship constant because it is so different from the "standard" house, software applications that are outside the "norm" of the types of applications developed in a single language will skew the backfiring constant. For a large village of fairly similar homes, the relationship between pallets of drywall and the number of square feet will provide a fairly consistent estimate of square feet. For a large number of similar software applications, the relationship between the FP and the SLOC by language also works fairly well. In both cases, the law of large numbers (the more data points in your sample size, the better the average results) usually provides a good overall portfolio approximation of the "village" size. The problems emerge when one tries to use a single data point with the translation constant to derive FP or SLOC from each other. This is similar to saying that the square feet in a house derived from the pallets of drywall will be accurate based on the equation relating the two measures. When a house (or a software application for that matter) does not fit the "norm" of the "average" in the data sample, there will obviously be variations in the accuracy of the result. A data entry system with fewer lines of code and a high language level will result in a lower number of FP, than one with many lines of code used to derive mathematical functions for use in reporting, even considering the effects of the language level. In these situations, the SLOC to FP ratio does not reflect the differences in logical functionality and will arrive at an inaccurate backfired FP count.[3]

Conclusions

At best, published tables of language translation factors should be viewed as no more than indicative, particularly when they are not qualified with details of their source, applicability and claimed accuracy. Like buying food without a "use by" date, it may be good, but the risks are unknown.

Software measurements are needed as the basis for project decision-making. And although the quality of the measurement needs to be no better than the decision that is to be based on it, it has to be acknowledged that bad information leads to bad decisions. For every measurement activity there needs to be a cost/benefit trade off. However, it is important to understand the uncertainties and risks associated with any measurement, and in the case of software size measures derived from backfiring there are significant uncertainties and risks. When backfiring you may get a quick, cheap measure, but also you get a crude risky measure.

The bottom line is that the use of backfiring is about on a par with estimates produced on the back of an envelope especially for single projects. The technique may be suitable for rough and ready calculations when the law of large numbers (and the large sample size) will even out the discrepancies between types of applications on a portfolio wide basis. However, backfiring, due to its inherent bias towards the physical nature of software, is simply not good enough as a basis for important project decisions that require an accurate functional size measurement.

About the Authors

Carol Dekkers is the President of Quality Plus Technologies, Inc., a U.S. based leading software measurement and process improvement consultancy. Carol has been involved in Function Point based measurement and outsourcing since 1994, and she is a frequent writer, presenter and trainer at major software quality conferences worldwide. Ms.

[3] A. Lubeshevsky of Lucent Technologies reported in a paper several years ago that the AT&T actual field tests of the backfiring technique resulted in a variation of up to 400% between the handcounted FP and the backfired FP number. Similar results have been experienced by the authors who have seen ratios of backfired FP figures to actual FP counts that have exceeded several hundred percent.

Dekkers was recently named one of the 21 New Faces of Quality for the 21st Century by the American Society for Quality (ASQ). She can be reached by e-mail at dekkers@qualityplustech.com

Ian Gunter is the President of Numerical Science, Inc., a London (England) based IT cost consultancy specializing in software estimating, outsourcing and process improvement. He is often retained by major businesses who have outsourced their systems development and/or maintenance activities and who are challenged to make the measurement provisions of their agreements work. Ian can be reached by e-mail at igg@numerical-science.com