

Evolving Function Points

Lee Fischman
Galorath Inc.

Functional size metrics for software emerged a generation ago with the invention of the function point. Since then, they have become the most common alternative to lines of code. Function points gauge software size in terms of delivered functionality rather than gross physical size, providing a valuable alternative perspective that often is preferred. Despite being a key innovation in software sizing, the software engineering community has not been entirely satisfied with function points. Consequently, alternative functional metrics (Mark II, Feature Points, and Full Function Points) have been proposed to remedy perceived deficiencies. This diversity, however, does not lead the software community to a standard that achieves widespread use. Moreover as the leading functional metric, function points deserve to be evolved rather than abandoned. This article outlines the findings of a metrics research program conducted during the last several years. The program explored function points' underlying framework, reviewed previous research, and considered changes to the current standard. The goal is to reconcile lingering criticisms of function points with the tremendous investment made in them during the past 20 years.

What do critics claim is wrong with function points? The critique below may be a long list, but hold your breath. It is not damning. Function points have been shown to be a definite indicator of development effort, and are still fundamentally sound.

Semantically Difficult. Function point standards were codified in the early 1980s by a standards body hailing from a traditional management information system world. Since then the standards document has not been drastically overhauled. Its language reflects this with seemingly arcane terms such as “record element types, external inputs, etc.” While such careful language insulates a relatively complex metric from everyday misunderstanding, it also impedes learning and acceptance by a wider audience.

Too Many Steps. The function point counting methodology is complex. It takes several days to learn function points, which is more time than most harried software engineers are willing to spend. Furthermore, some of that methodology is mathematically suspect while potentially adding no benefit.

Incomplete. Function points were defined from the user interface's vantage. Although a clever angle, this caused major criticism that all the functionality built into a software system might not be captured. Many argued that substantially internal functionality, without much manifestation at the user interface, might be missed.

Arbitrary Weightings. Once identified, *raw* function points go through two numeric transformations. The first is meant to weight them for relative size—low, average, high. The second is intended to make different types of points comparable such as equating an external input to an external output. The problem is that the scalar values behind these transformations were developed more than 20 years ago under very particular circumstances. At worst, these values may now be arbitrary.

No Automatic Count. No generally automated method is available for counting function points, even in completed systems. In contrast, lines of code counts can be obtained using simple line counting utilities. This paper does not address the automatic counting issue; innovations eventually may emerge from computer aided software engineering vendors.

Simple Semantic Changes

The following changes are intended to make function points easier to learn and eliminate inconsistencies.

Simpler Names. Function points' key innovation is that they approach software size from an intuitive perspective—user interface artifacts such as inputs, outputs, and files a software developer understands. Why call these *external inputs*, *external outputs* and *internal logical files* when more straightforward terms work equally well? Figure 1 offers a simplified nomenclature.

Figure 1. *Simplified Naming Scheme*

External Input	External Output	External Inquiry	Internal Logical File	External Interface File
↓				
Input	Output	Simple Input/Output	Internal Data Grouping	External Data Grouping

Simplified Weighting Terms. The function point methodology describes a function in terms of size. Actually, the *standard* refers to “complexity” but complexity is an algorithmic factor that should be orthogonal to a size metric, so we are unilaterally changing the label. Consistent with this change, low, average, and high complexity become *small*, *medium*, and *large*.

Size is determined by counting a function's attributes. The standard refers to these as data element types, record element types, and file types referenced—simpler terms are *field* for the first item and *data groupings accessed* for the latter two. Figure 2 illustrates how size is determined then labeled using the alternative nomenclature outlined here.

Accounting for Hidden Functionality

Function points are determined at an application's external

Figure 2. *Size Determination Matrix*

		Total Number of Fields		
		1 to 5	6 to 19	20 or more
Data Groupings Accessed	1	Small	Small	Medium
	2 to 3	Small	Medium	Large
	4 or more	Medium	Large	Large

interface, the layer where interaction with the outside world occurs. However, attributes at the external interface sometimes provide little indication of how substantial underlying code is. Examples include algorithmically intense software (encryption, image processing) or systems with underlying “layers” that are out of the user’s view. Judged from the external interface, the size of these systems will be understated. Two very different methods for capturing hidden size have been suggested but never before specified for use in a single framework.

An Internal Function Point. Numerous researchers have suggested a new function point to capture functionality missed by the other categories. It even has been implemented in competing functional metrics schemes. Figure 3 illustrates the idea behind the “internal function.”

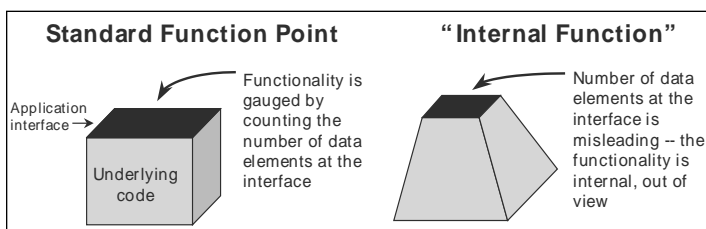


Figure 3. *The Internal Function “Iceberg”*

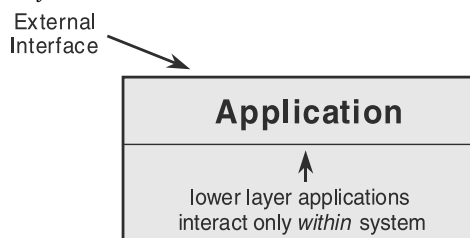
As depicted an internal function is a truly extraordinary input or output. It easily bests other functions that form an external perspective resembling it in size but have nowhere near the underlying amount of code. Keep in mind that these functions should occur rarely, no more than a few times in the average system.

When an internal function is found, it probably should be sized by analogy against standard function points. Compare an internal function against other known inputs or outputs in the system—it could equal *several*. Remember that an internal function is an input or output with a misleadingly simple external interface; sizing by analogy corrects this misjudgment.

Layers. Other hidden functionality can be captured by a change of perspective. A cornerstone of the function point framework is that software functionality, except key data structures, is not functional unless it interacts with the outside world. This external interface provides a consistent vantage while accounting for the entire system. However, this level can also conceal the inner workings of a complex system (see Figure 4).

Component-to-component interaction can be revealed with internal layers, an innovation first proposed for full function points. Beneath the external interface, layers are intended as equally valid perspectives from which to count function points. To prevent misinterpretation and overcounting, a layer must be strictly defined. All software has many functions interacting

Figure 4. *Layers*



with one another; these do not justify layers.

Internal layers are characterized by a well defined internal interface that every function in a system lies either above or below. They are tantamount to secondary application boundaries. Layers certainly exist when there are wholly constituted systems within systems, such as with middle-ware and operating system utilities. Inputs or outputs counted at each layer still must satisfy the counting rule that an internal file (data grouping) is modified.

Methodology Changes that Aid Learning

It takes several days to learn function point counting and more time to become proficient. This hurdle has limited the pool of trained counters. However, an exploration of the standard reveals potentially easier ways to learn to count.

Start from Artifacts. An alternative approach to learning function points is to start with a set of recognizable design artifacts and provide clarification only when necessary. This should be a much faster way to learn that establishes a critical intuitive link for skeptical software developers, the target audience. Figure 5 suggests mappings between artifacts and function points.

Functional Artifacts...	...Function Points
Input screens; Batch, interactive & hardware inputs	➔ Input
Reports; Media, software & hardware outputs	➔ Output
Simple inquiries; Other request/response	➔ Simple Input / Output
Functions with unrevealed but substantial underlying functionality	➔ Internal Function
Database tables; Long-lived groupings of data	➔ Internal Data Grouping
Read-only files; Outside tables	➔ External Data Grouping

Figure 5. *Mapping from artifacts*

Counting Rules Only as a Last Resort. Judging function points from artifacts is a shortcut that every experienced counter takes. However, a function is not a “point” unless specific rules are satisfied. These reinforce the formal framework behind function points and help to resolve discrepancies. The rules have been reformulated to make them slightly easier; these are not currently endorsed by any formal standards organization. The counting rules for transactional functions (inputs, outputs, input/output) can be reduced to three core rules that establish which observations constitute a point:

- Does this process leave the system in an equilibrium state? From the user’s perspective, this means that nothing is left to be done. The entire sequence of actions until a feature is satisfied should be considered part of a single point.
- Is this the smallest meaningful unit of activity? If adjacent pieces of functionality can work separately and each satisfy discrete functional requirements, then count them separately.
- Is the logic or data being handled unique to this process? If not unique, this functionality should not be counted.

The counting rules for files, recast in this article as data groupings, are:

- Is this group of data visible to the user via an input or output? Groupings of data are evaluated at the external interface

or internal layer (if you can accept the latter as an extension) and so they must naturally be evident there.

- Does this group of data logically belong together? If certain data items are always associated, then they belong in a single group. This reinforces the idea that function points are based on specifics of design rather than implementation. As such, physical attributes (tables, flat files, etc.) can delineate logical groupings of data.
- Has this group of data been counted before? A data grouping may be encountered in a system many times, but it only is designed (and counted) once.

The Math

Alongside the qualitative definition of function points there is a mathematical framework that is necessary for quantifying and summarizing them. Function points can be used for quantitative purposes such as for effort estimation only after they are transformed into a numeric value such as in effort estimation. Yet the standard methodology involves a loss in information and may be somewhat arbitrary.

Do not summarize into unadjusted function points. A crucial step in orthodox function point analysis is taking separately counted inputs, outputs, files, etc., and combining these into a single value, the *unadjusted function point count*. However, whether a function point is a file, input, or output is important information that is lost when function points are rolled into a single value. Function point counts by type should be retained so a maximum amount of information is available for later use.

If you are going to use weightings, be careful. Function points are counted by type and then weighted by size (see Figure 2). However, the weighting factors in common use were determined from IBM applications in the late 1970s. There is no proof they can be generalized to other organizations, technologies, and eras.

A few things can be done. First, accept the weightings. There is no alternative to them, and the standard weightings are required for comparison against third-party actuals databases. Alternatively, ignore the weightings and simply count by type, ignoring size. This approach could work if the function point count is used only for in-house purposes. A final option is to develop your own weighting scheme, perhaps backed up by another metric or by known effort relationships.

Conclusion

If every recommendation in this article were to be adopted, the result would be a function point standard that is markedly similar to the current one. The various simplifications proposed do not change counting results; meanwhile, extensions to account for hidden functionality would only rarely apply. Other suggestions are intended to increase the acceptance of function points and involve no changes to the underlying standard.

Functional size metrics are here to stay. As software technology continues to evolve, they eventually may be preferred to lines of code. The question is whether lingering concerns about function points will remain unanswered or whether many of the changes advocated here will be adopted.

Acknowledgments

Thanks to Alan Clark for editing assistance.

Epilogue

For a further understanding of function points, go to www.galorath.com/fp_tutorial. The internal function defined here is different from those previously proposed in that it must manifest at the user interface. The reason for this difference is the simultaneous provision for internal layers, which should capture truly internal functionality. If you have better ideas on how to size internal functions or how to transform a qualitative size scale to a numeric value, contact me. ♦

Recommended Readings

- Abran, Alain and Robillard, Pierre N., Function Points Analysis: An Empirical Study of Its Measurement Processes, *IEEE Transactions on Software Engineering*, Vol. 22, No. 12, pp. 895-910, December 1996.
- Albrecht, Allan J., Gaffney, John 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, pp. 639-648, Nov. 1983.
- Bock, Douglas B., Klepper, Robert, FP-S: A Simplified Function Point Count Method, *The Journal of Systems and Software*, July 1992, Vol. 18, No. 3, pp. 245-254.
- Briand, Lionel, El Emam, Khaled, and Morasca, Sandro *Theoretical and Empirical Validation of Software Product Measures*, Technical Report, Centre de Recherche Informatique de Montréal, Number ISERN-95-03, 1995.
- Fischman, Lee, *Analysis Of Function Point Rules In A Tree*, presented at the 1999 International Workshop On Software Metrics, available at www.galorath.com
- Fischman, Lee, *The Place of Function Points In An Underlying Model of Software Content*, presented at the 1999 IFPUG National Conference, available at www.galorath.com
- Fischman, Lee, *Function Point Counting For Mere Mortals*, presented at the 1999 Applications of Software Metrics Conference, available at www.galorath.com
- Harrison, Warren and Miluk, Gene. *The Impact of Within Size Variability on Software Sizing Models*, unpublished, available at www.galorath.com
- Ho, V.T., Abran, A., Oligny, S., *Using COSMIC-FFP to Quantify Functional Reuse in Software Development*, Escm-Scope 2000, available at www.lrgl.uqam.ca/ffp.html
- International Function Points User Group, *IFPUG Counting Practices Manual*, Version 4.1, www.ifpug.org
- Jones, Capers, *Feature Points (Function Point Logic for Real Time and System Software)*, presented at the fall 1988 IFPUG National Conference.
- Kemerer, Chris F., *Reliability of Function Points Measurement: A Field Experiment*, MIT Sloan School of Management. WP#3193-90-MSA.
- Kitchenham, Barbara and Pfleeger, Shari L., and Fenton, Norman *Towards a Framework for Software Measurement Validation*, *IEEE Transactions on Software Engineering*, 21(12), pp. 929-943, December 1995.
- Oligny, Serge and Abran, Alain, *On the Compatibility Between Full Function Points and IFPUG Function Points*, unpublished, available at www.uqam.ca
- Symons, Charles R., *Software Sizing and Estimating Mk II Function Point Analysis*, John Wiley & Sons, 1991.

About the Author



Lee Fischman is a special projects director at Galorath Incorporated and is responsible for applications development, design, and research projects. Fischman is also a frequent speaker at national conferences. He studied economics at the University of Chicago and UCLA.

Galorath Incorporated
100 North Sepulveda Boulevard, Suite 1801
El Segundo, Calif. 90245
Voice: 310-414-3222
Fax: 310-414-3220
E-mail: info@galorath.com

QUOTE MARKS

Software is like Entropy: It's hard to grasp, weighs nothing, and obeys the second law of thermodynamics, i.e. it always increases.

— Norman Augustine

"A language that doesn't affect the way you think about programming is not worth knowing."

— Anonymous

"Man invented language to satisfy his deep need to complain."

— Lily Tomlin

Coming Events

February 7-9

Network and Distributed System Security Symposium
www.isoc.org/ndss01/call-for-papers.html

February 12-16

Software Management Conference
www.sqe.com/sm



February 12-16

Applications of Software Measurement Conference
www.sqe.com/asm

March 5-8

Software Testing Analysis and Review
www.sqe.com/stareast

March 5-8

Mensch and Computer 2001
<http://mc2001.informatik.uni-hamburg.de>



March 12-15

SEPG 2001
FOCUSING ON THE DELTA
Software Engineering Process Group Conference
www.sei.cmu.edu/products/events/sepq


March 31-April 5

Conference on Human Factors in Computing Systems
www.acm.org/sigs/sigchi/chi2001

April 22-26

 *Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*
www.ieee-infocom.org/2001 

April 29-May 4

 *Software Technology Conference (STC 2001)*
www.stc-online.org

May 1-3

2001 IEEE Radar Conference
www.atlaessgrss.org/radarcon2001

May 12-19

23rd International Conference on Software Engineering, and International Workshop on Program Comprehension
www.csr.uvic.ca/icse2001

June 03-06

2001 IEEE Southwest Test Workshop
E-mail: william.mann@ieee.org

June 11-13

E-Business Quality Applications Conference
<http://qaiusa.com/conferences/june2001/index.html>

June 18-22

ACM/IEEE Design Automation Conference
www.dac.com



June 25-27

2001 American Control Conference
www.ece.cmu.edu/~acc2001