

Function Point Software Sizing

by

Anthony A. DeMarco
Director, PRICE Systems

Introduction

The number of source lines of code (SLOC) has consistently proven to be the most reliable sizing metric to estimate software development and support costs. However, it is difficult to determine SLOC for a conceptualized system without sound structured analysis of an extensive experience base. Thus, many software sizing methods have been invented and refined over the past quarter century to address the dilemma. The most popular technique, by far, is *function point* sizing.

In the mid 1970's, Allan J. Albrecht of IBM was tasked with measuring software productivity. He needed a method of measuring the size of a software development task independent of programming language and environment. In 1979, Albrecht published a paper¹ in which he first described a software sizing metric termed *function points*. During the early 1980's function point metrics and function point counting procedures were employed by IBM and others with some initial success. In 1983, Albrecht and John E. Gaffney, Jr.² refined the function point technique by expanding function type definitions and counting procedures.

What are Function Points?

Briefly, function point sizing involves counting five different categories of functions in a software application: *inputs, outputs, inquiries, interfaces, and internal files*. These are functions the *user* of the application can see and identify. Each function is qualified in terms of complexity (Low, Average, or High) and then multiplied by a corresponding complexity weight to achieve a *function point count*, a measure of the size of the function. The sum of function point counts for the functions is called the total *unadjusted function point count* (UFPC). Next, fourteen general system characteristics are rated with respect to *degree of influence*, zero through five with zero indicating no influence and five indicating an extremely strong influence. The characteristics include qualities such as data communications, performance, and operational ease. The sum of the degrees of influence for the fourteen characteristics is used to make a *value adjustment* to the UFPC. The result is the *total function point count* for the application.

IFPUG

Function point counts as a measure of software size was in wide use by the mid-1980's. But function point counters' definitions and practices varied just as widely - making industry-wide comparisons meaningless. In 1986, the International Function Point Users Group (IFPUG) was founded to standardize function type definitions and counting practices. Today IFPUG has over 500 members representing more than 400 different companies. They publish a "Function Point Counting Practices Manual"³, hold semi-annual conferences, and disseminate information to promote the use of the function point process.

Function Point Counting

Before you begin counting function points, the concept of *application boundary* must be understood (Figure 1). The application boundary separates the application from the user and other interfacing applications. Functions are either *internal* or *external* to the application boundary. External functions cross the boundary while internal functions stay within the boundary. Inputs, outputs, inquiries and

interfaces are external, logical files are internal. Hence, the first step in function point analysis is to define and visualize the boundary around the application.

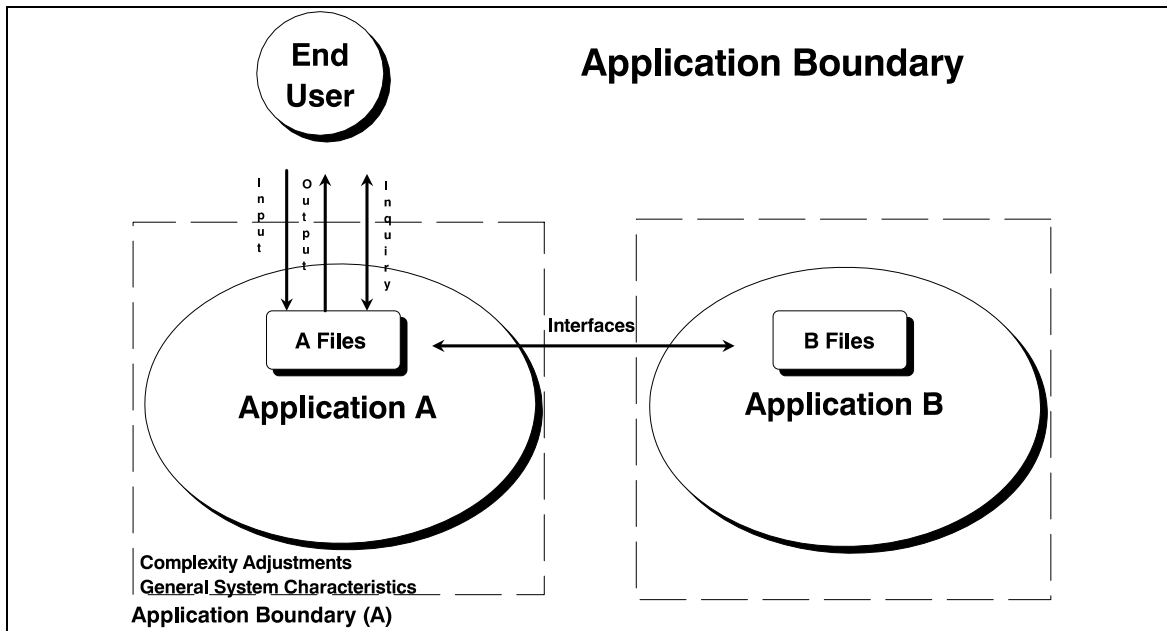


Figure 1: Application Boundary

Next, you identify functions by type. Here the IFPUG manual comes in very handy. Each function type and its corresponding complexity is defined in great detail and illustrated with examples and counter-examples. IFPUG defines the five function types:

Internal Logical File (ILF)

An *Internal Logical File* is a user identifiable group of logically related data or control information maintained and utilized within the boundary of the application. Internal logical files represent an application's maintainable data storage requirements.

External Input (EI)

An *External Input* processes data or processes control information which enters the application's external boundary. The processed data, through a unique logical process, maintains an Internal Logical File. Control information is data used by a process within an application boundary to assure compliance with business function requirements specified by the user. Control information may or may not directly maintain an Internal Logical File. An External Input should be considered unique if it has a different format or if the logical design requires processing logic different from other External Inputs of the same format. External Inputs represent an application's data maintenance and control processing requirements. An External Input is considered unique if data is maintained on an Internal Logical File and the input format is unique or the processing logic is unique.

External Output (EO)

An *External Output* processes data or control information that exits the application's external boundary. An External Output should be considered unique if it has a different format, or if the logical design requires processing logic different from other External Outputs of the same format. External Outputs represent an application's output processing requirements. An external output is considered unique if the output format is unique or the processing logic is unique.

External Interface File (EIF)

An *External Interface File* is a user identifiable group of logically related data or control information utilized by the application but maintained by another application. External Interface Files represent an application's externally maintained data storage requirements.

External Inquiry (EQ)

An *External Inquiry* is a unique input/output combination that results in the retrieval of data required for immediate output. It does not contain derived data and does not update an Internal Logical File. An External Inquiry is considered unique if it has a format different from other External Inquiries in either its input or output parts or if the logical design edits and sorts different from other External Inquiries. External Inquiries represent an application's inquiry processing requirements. An input/output combination is considered unique if the input format is unique or the edits and/or sorts are different or the output format is unique.

Do these functions sound foreign to you? Consider an encounter with your friendly Automated Teller Machine (ATM):

- 1. You insert your ATM card *...1 External Input*
 - 2. You enter your PIN number *...another External Input*
 - 3. You request an on screen account balance *...1 External Inquiry*
 - 4. You enter a withdraw amount *...another External Input*
 - 5. You receive cash and a receipt *...2 External Outputs*
- ...and the entire time a number of External Interface Files are being queried to access your account data*

Get the idea?

After you identify an application's functions, you determine each function's complexity. Again, the IFPUG manual is very specific. Below is a table used to assign complexity to an external input.

External Input Complexity Matrix			
	1 to 4 DET	5 to 15 DET	16 or more DET
0 to 1 FTR	Low	Low	Avg.
2 FTR	Low	Avg.	High
3 or more FTR	Avg.	High	High
where: FTR = File Type Referenced DET = Data Element Type			

Table 1: External Input Complexity Matrix

Now multiply function counts by corresponding complexity weights (Table 2) to compile an *unadjusted function point count* (UFPC).

Function Type	Functional Complexity	Complexity Totals	Function Type Totals
Internal Logical Files	Low	x 7 = _____	_____
	Average	x 10 = _____	
	High	x 15 = _____	
External Interface Files	Low	x 5 = _____	_____
	Average	x 7 = _____	
	High	x 10 = _____	
External Inputs	Low	x 3 = _____	_____
	Average	x 4 = _____	
	High	x 6 = _____	
External Outputs	Low	x 4 = _____	_____
	Average	x 5 = _____	
	High	x 7 = _____	
External Inquiries	Low	x 3 = _____	_____
	Average	x 4 = _____	
	High	x 6 = _____	
Unadjusted Function Point Count			_____

Table 2: Unadjusted Function Point Count Calculation.

The next step is to determine the application's *processing complexity*. You do this by judging the degree of influence imposed by fourteen general system characteristics (Table 3). Each characteristic is rated from zero (no influence) through five (extremely strong influence) and then the ratings are summed to arrive at the *total degree of influence* (TDI).

General System Characteristics	
1. Data Communication	8. On-Line Update
2. Distributed Processing	9. Complex Processing
3. Performance	10. Usable in Other Applications
4. Heavily Used Configuration	11. Installation Ease
5. Transaction Rates	12. Operational Ease
6. On-Line Data Entry	13. Multiple Sites
7. Design for End User Efficiency	14. Facilitate Change

Table 3: General System Characteristics

Finally, the total *function point count* (FPC) is determined by the equation below:

$$\text{FPC} = \text{UFPC} \times [(\text{TDI} \times 0.01) + 0.65]$$

Function Points and SLOC

SLOC is the most reliable sizing metric from which to estimate software development and maintenance efforts. Since SLOC can be physically counted, it is easier to establish from past programs than function points. But function points are more comfortable to work with during early conceptual phases. Hence, as function points grow in popularity, interest grows in correlating function point counts to SLOC^{2,4}. Strong correlation has led to programming language factors which can be used to translate between function points and SLOC. These findings make function point sizing a viable means of determining SLOC for conceptualized software systems.

Function Point Sizing in PRICE S

The latest release of PRICE S offers a sizing utility which employs function point counting to estimate SLOC, Function Point Sizer. An example from Dreger⁵ will be used to illustrate Function Point Sizer.

On-Line Parts System

A batch system exists for parts inventory control. An on-line, menu driven system is proposed to supplement the batch system. The desired capabilities include:

- on-line request for reports
- on-line display of parts description
- on-line display of parts inventory
- on-line file maintenance

Only a portion of the Master Parts File will be contained in a Selected-Parts File to speed processing. The parts contained in the Selected-Parts File are determined by the users of the on-line system. Control and inventory reports will be available. The system flowchart and corresponding function counts are shown in Figure 2.

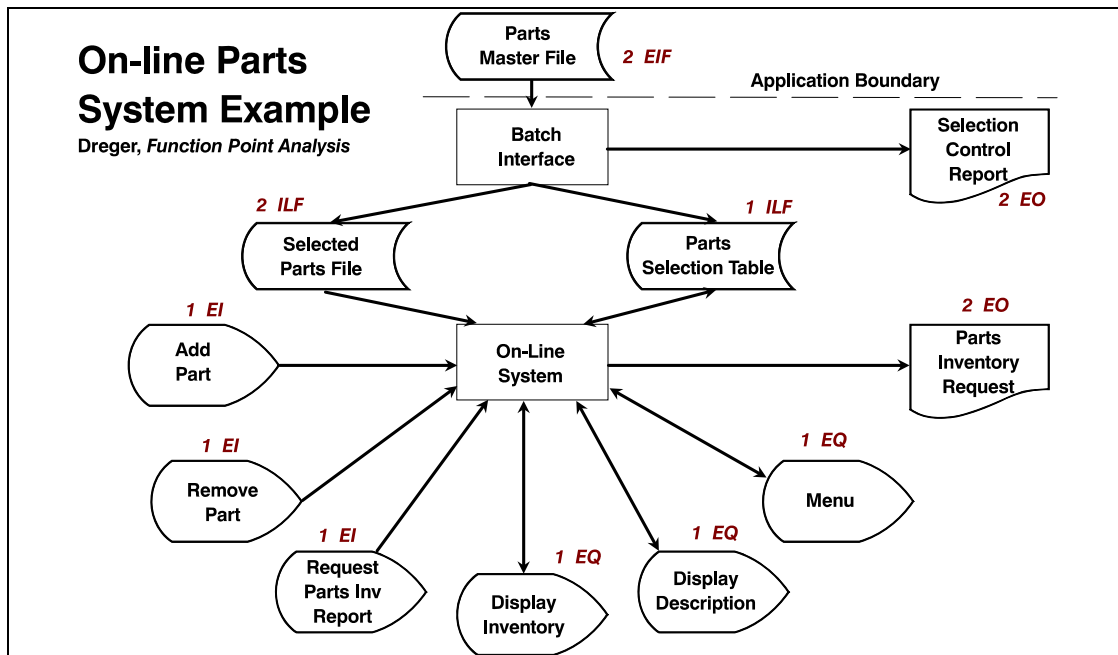


Figure 2: On-Line Parts System Flowchart

The dialog box below shows the On-line Parts System modeled with PRICE S's Function Point Sizer.

Function Point Sizing					
Unadjusted Function Point Calculation				Value Adjustment Factor Calculation	
Function Type	Functional Complexity	Complexity Totals	Function Type Totals	General System Characteristics	Degree of Influence
Internal Logical Files	<input type="text" value="3"/>	Low * 7 = 21	21	1. Data Communications	<input type="text" value="3"/> <input type="button" value="↓"/>
	<input type="text" value="0"/>	Avg. * 10 = 0		2. Distributed Processing	<input type="text" value="0"/> <input type="button" value="↓"/>
	<input type="text" value="0"/>	High * 15 = 0		3. Performance	<input type="text" value="5"/> <input type="button" value="↓"/>
External Interface Files	<input type="text" value="2"/>	Low * 5 = 10	10	4. Heavily Used Configuration	<input type="text" value="5"/> <input type="button" value="↓"/>
	<input type="text" value="0"/>	Avg. * 7 = 0		5. Transaction Rates	<input type="text" value="4"/> <input type="button" value="↓"/>
	<input type="text" value="0"/>	High * 10 = 0		6. On-Line Data Entry	<input type="text" value="3"/> <input type="button" value="↓"/>
External Inputs	<input type="text" value="3"/>	Low * 3 = 9	9	7. Design for End User Efficiency	<input type="text" value="2"/> <input type="button" value="↓"/>
	<input type="text" value="0"/>	Avg. * 4 = 0		8. On-line Update	<input type="text" value="3"/> <input type="button" value="↓"/>
	<input type="text" value="0"/>	High * 6 = 0		9. Complex Processing	<input type="text" value="1"/> <input type="button" value="↓"/>
External Outputs	<input type="text" value="2"/>	Low * 4 = 8	18	10. Usable in Other Applications	<input type="text" value="0"/> <input type="button" value="↓"/>
	<input type="text" value="2"/>	Avg. * 5 = 10		11. Installation Ease	<input type="text" value="2"/> <input type="button" value="↓"/>
	<input type="text" value="0"/>	High * 7 = 0		12. Operational Ease	<input type="text" value="5"/> <input type="button" value="↓"/>
External Inquires	<input type="text" value="0"/>	Low * 3 = 0	12	13. Multiple Sites	<input type="text" value="5"/> <input type="button" value="↓"/>
	<input type="text" value="3"/>	Avg. * 4 = 12		14. Facilitate Change	<input type="text" value="4"/> <input type="button" value="↓"/>
	<input type="text" value="0"/>	High * 6 = 0		Total Degree of Influence	42
Unadjusted Function Point Count			70	Translation <input type="text" value="105.0"/> SLOC 7770	
Function Point Count			74	<input type="button" value="OK"/> <input type="button" value="Cancel"/> <input type="button" value="Military Sizer"/> <input type="button" value="Commercial Sizer"/>	

Figure 3: Function Point Sizer

Function Point Sizer is reachable from any of the model's SLOC input fields. The utility has four basic components: the Unadjusted Function Point Calculation, the Value Adjustment Factor Calculation, the Translation Factor Table, and the computed SLOC. In this case, function counts and complexity ratings are determined from the On-Line Parts System's concept document. For example, the *Add Part* function is an external input: *the function processes control information which enters the application's boundary*. The Add Part screen (envisaged as in Figure 4) requires the user to enter two inputs: *part number* and *size code*. From Table 1, this external input has a *Low* complexity rating.

On-Line Parts System

Add New Part

Please Enter: Part # _____

 Size Code _____

Figure 4: Add Part Screen

Function Point Sizer reacts to every keystroke. You enter function type counts in each of the five categories by complexity ranking and the utility instantly computes function point counts by type and the total UFPC. Each input parameter is well documented by the PRICE S context sensitive, hypertext help system. As the degree of influence is determined for each system characteristic, the UFPC is automatically adjusted and the total Function Point Count is displayed. You can see from the example that the On-Line Parts System represents an UFPC of 70, which is adjusted by the 42 TDI to get 74 function points.

Finally, you enter a translation factor or choose one from the pop-up table (Figure 5) to achieve a SLOC count for the software item. The On-Line Parts System will be programmed in Cobol which is typically 105 lines of code per function point. The 74 function points is multiplied by 105 to achieve 7,770 Cobol source lines for the application.

Function Point to SLOC Translation Table	
<input type="button" value="OK"/> <input type="button" value="Cancel"/>	
Translation <input type="text" value="105.0"/>	
<input type="radio"/> Ada..... 71	<input checked="" type="radio"/> Cobol..... 105
<input type="radio"/> Algol..... 105	<input type="radio"/> Compass..... 492
<input type="radio"/> Apl..... 32	<input type="radio"/> Coral-66..... 64
<input type="radio"/> Assembly..... 320	<input type="radio"/> Flod..... 32
<input type="radio"/> Atlas..... 32	<input type="radio"/> Fortran..... 58
<input type="radio"/> Basic..... 64	<input type="radio"/> Ifam..... 25
<input type="radio"/> C..... 128	<input type="radio"/> Jovial..... 105
<input type="radio"/> CMS-2..... 178	<input type="radio"/> Microcode..... 107
<input type="radio"/> Pascal..... 91	<input type="radio"/> PL1..... 80
<input type="radio"/> PL1..... 80	<input type="radio"/> Pride..... 64
<input type="radio"/> Pride..... 64	<input type="radio"/> SPL1..... 291
<input type="radio"/> SPL1..... 291	<input type="radio"/> High-Order..... 105
<input type="radio"/> High-Order..... 105	<input type="radio"/> Machine..... 320
<input type="radio"/> Machine..... 320	<input type="radio"/> 4th-Generation..... 15
<input type="radio"/> 4th-Generation..... 15	<input type="radio"/> Interpretive..... 64
<input type="radio"/> Interpretive..... 64	

Figure 5: Function Point to SLOC Translation Table

When you are finished sizing the application, the estimated SLOC along with other descriptive parameters is used by the model to estimate the development and support costs and schedules. Figure 6 shows that PRICE S estimates 20.8 person-months over 7.9 calendar months to develop the On-Line Parts System. The true power of including Function Point Sizer in PRICE S is the ability to perform rapid *What-If?* analysis. You simply add or delete functions and see the impact on program cost. Change the degrees of influence and immediately view the impact on monthly expenditures. Design-to-cost tradeoffs can be analyzed instantly .

Biography

Anthony A. DeMarco

Mr. DeMarco is Director of PRICE Systems in Moorestown, New Jersey. He leads a team of operations researchers, software developers, and cost analysts who create and support computer based parametric cost analysis and estimating tools.

Mr. DeMarco received a Bachelors degree in Mathematics from St. Joseph's University in Philadelphia, Pennsylvania, and a Masters degree in Computer Science from the New Jersey Institute of Technology.

Since joining PRICE Systems in 1981, Mr. DeMarco's accomplishments have included the development of the PRICE electronics cost model, PRICE M, major enhancements to the PRICE hardware model, PRICE H, and the PC program XPERT/H. Mr. DeMarco has written and presented many papers on parametric modeling and he has published articles in the RCA Engineer and the NES Estimator. He can be reached at:

PRICE Systems
300 Route 38
Moorestown, NJ 08057
1-800-43-PRICE

¹**Allan J. Albrecht**, "Measuring Application Development Productivity", *Proceedings, Joint SHARE/GUIDE/IBM Application development Symposium*, October 1979, pp. 83-92.

²**Allan J. Albrecht and John E. Gaffney, Jr.**, "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation", *IEEE Transactions on Software Engineering*, pp. 639-648, November 1983.

³**International Function Point Users Group**, "Function Point Counting Practices Manual", IFPUG, 5008-28 Pine Creek Drive, Westerville, Ohio 43081-4899.

⁴**T. Capers Jones**, "Function-Point Metrics: Key to Improved Productivity", *Information Week*, pp.26-27, 23 February 1987.

⁵**J. Brian Dreger**, "Function Point Analysis", Prentice Hall, Englewood Cliffs, New Jersey, 1989.