APPENDIX A

An Introduction to Function Point Analysis

This appendix provides more information about function point analysis. Keep in mind that even this discussion will provide you with only a basic understanding. Although function point analysis is not difficult, the rules for counting function points can be complex for the novice. Resources, such as books, Web sites, training, and certification, are widely available if you are interested in learning more.

BACKGROUND

Lines of code (LOG) or source lines of code (SLOC) have been the traditional way of estimating the size of an application. Although intuitively appealing, estimating or counting lines of code have several disadvantages. First, many organizations develop applications using different programming languages, platforms, tools, and so on. An IT project developed in Visual Basic and SQL Server will be difficult to compare to a mainframe-based COBOL application. Moreover, experienced and talented programmers tend to write more efficient code than novice programmers. As a result, experienced programmers may write fewer lines of code than novices and still accomplish the same thing. In addition, no set standard exists for determining what exactly should be counted. For example, should remarks or documentation lines be counted? What about the initialization of variables? Although counting lines of code seems fairly straightforward, the actual implementation becomes problematic.

To overcome many of the inherent problems with counting LOG, Allan Albrecht proposed the idea of function points at a conference sponsored by IBM in 1979. The basic concept behind function points is to focus on *the functionality* of the application. After all, the size and complexity of an application (and subsequently the number of lines of code to be written) are based upon what the application must do. Function points provide a synthetic metric, similar to hours, kilos, and degrees Celsius, for software engineering that gives consistent results, regardless of the technology or programming language used.

In the early 1980s, statistical analysis provided the means for refining the function point technique. Since 1986, function point analysis rules and guidelines have been overseen by a nonprofit organization called the International Function Point Users Group (IFPUG). The IFPUG maintains the *Counting Practices Manual* that contains all the current guidelines and certification for counting function points under the IFPUG standard. The material in this appendix will be based upon the latest counting practices by IFPUG.

You should know, however, that there is an alternative way of counting function points. In 1983, Charles Symons, working for Nolan, Norton, and Company (later acquired by KPMG Consulting) critiqued Albrecht's proposed function point technique and argued the existence of several flaws. As a result, Symons proposed an alternative function point technique called the Mark II approach. The Mark II technique has become popular primarily in the United Kingdom and is overseen by the United Kingdom Function Point Users Group (UFPUG).

WHAT PRECISELY IS A FUNCTION POINT?

Function point analysis is a structured technique for breaking up or modularizing an application by categories or classes based on functionality. A function point is a software metric. Similar to the many metrics you use each day, a function point provides an idea of the size and complexity of a particular application or module of that application. For example, it should be pretty straightforward that a 4,000-square-foot home is larger than a 2,000-square-foot home. But will a 4,000-square-foot house take twice as long and cost twice as much as a 2,000-square-foot house? It depends. What if the larger house uses stock material and includes only the basic amenities while the 2,000-square-foot house has many custom features? The custom features may include a handcrafted staircase, exotic wood, imported marble, and other very expensive items. As you can see, depending on the features or requirements of each house, the time to build and the cost for each house can differ radically (Dekker 1999).

Similarly, an application that has 4,000 LOG has twice as many lines of code as a 2,000 LOG application. But will a 4,000 LOG application take twice as long and cost twice as much to build as a 2,000 LOG application? Again, the answer is that it depends. In this case, it depends more on the features or required functionality of the system and the complexity of those required features. Function points provide a useful metric that combines both functionality and complexity. For example, a 4,000 function point application will, in fact, be larger, have more functionality, and be more complex than a 2,000 function point application. Since function points are independent of the technology, we can compare these two applications regardless of the fact that one application is written in Java and the other in COBOL. More specifically, the size of the application is based upon functionality in terms of:

- Inputs
- Outputs
- Inquiries
- Internal files
- External files
- The complexity of the general characteristics of the system

Therefore, the key to function point analysis is having a good understanding of the system's requirements. Often at the outset of a project, the requirements may not be clear. A function point analysis can still be conducted and then updated throughout the project life cycle as these requirements become more clearly defined. For example, a function point analysis can be conducted based upon the definition of the project's scope. This analysis will provide a solid definition of the application's boundary and will provide a starting point for defining and subsequently estimated the size and complexity of the application deliverable. A clearer picture of the features and functionality of the application will follow during the analysis and design phases of the project. Later on, a function point analysis can be conducted when the project application is delivered, in order to compare the agreed upon requirements to what was delivered. In general, function points can be useful for:

- *Managing Scope*—Scope changes will change an application's total func tion point count. As a result, the project manager and project sponsor/client use function point analysis to determine the impact of a proposed scope change in terms of the project's schedule and budget.
- *Benchmarking*—The value of function point analysis is that data can be col lected and compared to other projects. For example, the true value of counting function points is to compare a project to past projects and to other projects throughout the organization. This comparison allows an organization to identify challenges and opportunities in order to take cor rective action when necessary. In addition, estimation becomes more mean ingful and accurate when similar methods, tools, and resources are part of the data analysis. An organization can inventory its application portfolio to understand cost structures and the impact of new best practices. Function points by themselves do not provide much information without the use of other metrics, such as time, cost, and quality.
- *Reliability*—Once knowledgeable and experienced in function point count ing, different people can count function points for the same application and obtain the same measure within an acceptable margin of error.

HOW TO CONDUCT A FUNCTION POINT ANALYSIS

The process of conducting a function point analysis can be summarized in seven steps:

- Determine the function type count to be conducted.
- Define the boundary of the application.
- Define all data functions and their degree of complexity.
- Define all transactional functions and their complexity.
- Calculate the Unadjusted Function Point Count.
- Calculate the Value Adjustment Factor based on a set of General System Characteristics.
- Calculate the final Adjusted Function Point Count.

Step 1: Determine the Function Type Count to Be Conducted

The first step in conducting a function point analysis is to determine the type of function count to be conducted. Function points can be counted by an individual or a small team, and the type of function point count will help the counters plan their strategy and determine what documents and resources will be required. A function type count can be one of three types:

- *Development*—A development function type count would be made for a new project. These types of counts would be based initially on the scope definition of the project and would be updated throughout the project life cycle as requirements and functionality are more clearly defined. The basic purpose of development function type counts is estimating the size and effort of the application.
- *Enhancement*—Enhancement focuses more on maintenance projects, or projects that attempt to modify or enhance existing applications. These

projects may include deleting, changing, or adding functionality to the existing application.

- *Application*—An application function type count may be viewed as an inventorying of an existing application in the IT project portfolio in order to create a baseline or benchmark. Combined with other metrics, a database can be created to support analysis and estimation.

Step 2: Define the Boundary of the Application

The application boundary defines the border for the user, the application itself, and any other external application. The boundary should be based upon the user's view of the domain and not technology partitions or platforms. Often applications today must interface or integrate with each other, so it is important that the boundary be defined clearly. Scope management is concerned with defining, managing, and controlling the project's scope. More specifically, tools such as data flow diagrams and use case diagrams are useful for defining the project's scope and the boundary for the application.

Step 3: Define All Data Functions and Their Degree of Complexity

Data function types may be thought of as data at rest; they are the logical data that can be updated and queried. The transactional functions, such as external inputs (El), external outputs (EO), and external inquiries (EQ), are processes that set the data in motion. These processes act directly on the logical data to perform the updates and queries. In particular, data functions can be either internal logical files (ILF) or external interface files (EIF). As their names imply, ILFs are maintained within the application boundary and EIFs are maintained by an external application but available to the application being counted. For example, a sales application might keep track of customers and the products they purchase, but customer balances and other credit-related information may be maintained by a separate accounts receivable application.

Once the ILFs and EIFs are identified and counted, they are scored or rated based on their functional complexity in terms of their number of record type elements (RETs) and data element types (DETs). A record type element, or RET, is a recognizable subgroup of data elements contained within the ILF or EIF. These are one of the more difficult concepts in function point analysis, but you can think of them as representing a parent-child relationship. In object-oriented terms, you can think of this as a subclass and a superclass. On the other hand, a data element type, or DET, is defined as a unique, non-recursive field recognized by the user. For example, let's say that an entity called student has a student identification number, name, address, and a cumulative number of credit hours. In addition, there are two types of students—undergraduate and graduate. If the data about students were stored, updated, retrieved, and queried by our application, we would count this as 1 ILF with 6 DETs and 2 RETs as illustrated in Table A. 1.

Once the ILFs and EIFs and their associated RETs and DETs have been identified and counted, their complexity can be determined using the matrix shown in Table A.2. For example, the Student ILF would have a complexity score of Low because the number of RETs is less than 2 and the number of DETs is between 1 and 19.

Step 4: Define all Transactional Functions and Their Complexity

Transactional functional types focus on the processing of data between the user and the application and between the application and any external applications. Therefore, transactional functions, called external inputs (Els), external outputs (EOs), and external inquiries, (EQs) perform updates, retrievals, and queries on the data contained within the ILFs and ElFs.

An external input (El) is defined as an elementary process that processes data or control information that originates from outside the application boundary. An elementary process is defined as the smallest unit of activity that is meaningful to the user. The elementary process must be viewed from the user's perspective (i.e., not a technical perspective) and must leave the application in a consistent state after performing its function. Data refers to the actual data processed by the transaction, while control information refers to such things as rules or parameters passed to application. An example of an El would be an input screen to add new students to the student ILF. The elementary process would require that all required fields be filled before adding the new student's information to the student ILF in order to leave the application in a consistent state.

Table A.1 Data Function Count

	Count As
Superclass: Student	ILF
Student ID number	DET
Name	DET
Address	DET
Cumulative credit hours	DET
Subclass: Graduate	RET
Graduate assistantship	DET
Subclass: Undergraduate	RET
Class standing	DET

Once the Els have been identified and counted, their complexity can be determined using the following matrix based on the file types referenced (FTR) and data element types. An FTR is just the number of ILF and EIF files referenced. For example, if an input screen to add new students only accessed the student ILF and included only 6 DETs, the complexity rating for this particular El would be Low. See Table A.3.

Similarly, an external output (EO) is an elementary process that allows data or control information to exit the application boundary. Examples of EOs would include reports, receipts, confirmation messages, derived or calculated totals, and graphs or charts. Once the EOs are identified and counted, their relative functional complexity can be determined based on the FTRs and DETs. Continuing with our example, suppose that the student application printed two reports, one report listing all the students alphabetically and the other grouping by graduate and undergraduate. If all data fields were included in each report, the complexity rating for the application's EOs would be Low See Table A.4.

Table A.2 Complexity for ILFs and EIFs

	DET: Data Element Type		
RET: Record Element Type	*1–19*	*20–50*	*51 or More*
Less than 2	Low	Low	Average
2–5	Low	Average	High
More than 5	Average	High	High

Table A.3 Complexity for External Inputs (EI)

	DET: Data Element Type		
FTR: File Types Referenced	*1–4*	*5–15*	*16 or More*
Less than 2	Low	Low	Average
2	Low	Average	High
More than 2	Average	High	High

Table A.4 Complexity for External Outputs (EO)

	DET: Data Element Type		
FTR: File Types Referenced	*1–5*	*6–19*	*20 or More*
Less than 2	Low	Low	Average
2–3	Low	Average	High
More than 3	Average	High	High

An external inquiry (EQ) is defined as an elementary process that includes both a combination of inputs and outputs for retrieving data from one or more ILFs and/or EIFs. Unlike an El, the EQ input process does not update any internal or external files, and the output of the EQ transaction does not calculate or derive any data. Once the EQs have been identified and counted, a relative complexity score can be made. For example, let's suppose our student application allows searching by student number. This query would count as one EQ. In addition, let's suppose that an error message is displayed if no matching student numbers are found. The number of DETs would include the 6 data fields plus an additional DET for the error message. Therefore, the complexity rating for the application's EQ would be Low. See Table A.5.

Step 5: Calculate the Unadjusted Function Point Count

Using the counts for each ILF, EIF, El, EO, and EQ, an Unadjusted Function Point count can be computed using Table A.6.

To find the Total Unadjusted Function Point Total (UAF), multiply the number of low, average, and high ILFs, EIFs, Els, EOs, and EQs by the appropriate number in each cell. These values are then summed across the rows for each function type. The grand total is just a summation of these row totals.

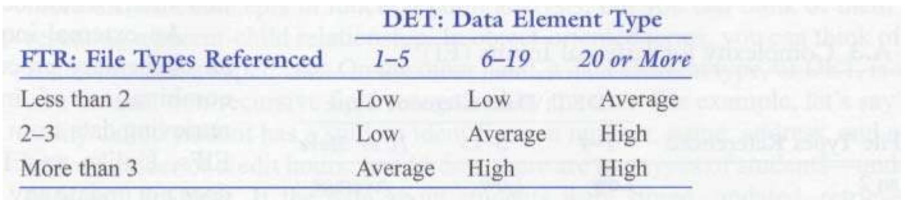
Step 6: Calculate the VAF Based on a Set of General System Characteristics

The Value Adjustment Factor (VAF) is multiplied by the Unadjusted Function Point (UAF) calculated in step 5 to come up with a Final Adjusted Function Point total. In identifying each ILF, EIF, EO, El, and EQ, a complexity matrix was used to determine the complexity for each data and transactional function type in terms of low, average, or high complexity. However, at this time a set of fourteen General System Characteristics (GSC) are used to compute a Total Degree of Influence. This degree of influence will be used to compute the VAF.

To determine the Total Degree of Influence, each GSC is rated based on its degree of influence using the following 0 to 5 scale:

0. Not present or no influence
1. Incidental influence
2. Moderate influence
3. Average influence
4. Significant influence
5. Strong influence throughout

Table A.5 Complexity for External Inquiries (EQ)



	DET: Data Element Type		
FTR: File Types Referenced	*1–5*	*6–19*	*20 or More*
Less than 2	Low	Low	Average
2–3	Low	Average	High
More than 3	Average	High	High

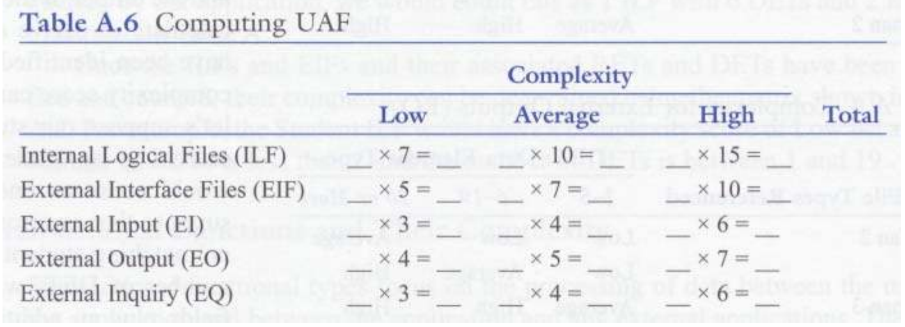


Table A.6 Computing UAF

	Complexity			
	Low	Average	High	Total
Internal Logical Files (ILF)	__ × 7 = __	__ × 10 = __	__ × 15 = __	
External Interface Files (EIF)	__ × 5 = __	__ × 7 = __	__ × 10 = __	
External Input (EI)	__ × 3 = __	__ × 4 = __	__ × 6 = __	
External Output (EO)	__ × 4 = __	__ × 5 = __	__ × 7 = __	
External Inquiry (EQ)	__ × 3 = __	__ × 4 = __	__ × 6 = __	

Total Unadjusted Function Points (UAF)

Following is information about each GSC that can be used to rate it.

1. *Data Communications*—A communication facility is required to send data and control information via teleprocessing (TP). These links require protocols that allow for the exchange of data between a sender and receiver. Examples include TCP/IP, Ethernet, AppleTalk, etc.

   Degree of Influence

   0. Pure batch or stand-alone PC
   1. Batch but with remote data entry or printing
   2. Batch but with remote data entry and remote printing
   3. Online data collection or TP on the front end to a batch processing or query system
   4. More than a front end, but only one type of TP protocol supported
   5. More than a front end with more than one type of TP protocol sup ported

2. *Distributed Data Processing*—Distributed data processing is a character istic of the application.

   Degree of Influence

   0. Does not aid the transfer of data or processing function between com ponents of the system
   1. Prepares data for end user processing or another component of the sys tem (e.g., spreadsheet, DBMS, etc.)
   2. Data prepared for transfer, then transferred and processed by another component
   3. Distributed processing and data transfer are online but only in one direction
   4. Distributed processing and data transfer are online and in both direc tions
   5. Processing of functions is dynamic and performed by the most appro priate component of the system

3. *Performance*—Performance in terms of response time or throughput. It will greatly influence the design, development, implementation, support, and maintenance of the application.

   Degree of Influence

   0. No special performance requirements stated
   1. Performance and design requirements stated and reviewed, but no spe cial attention needed
   2. Response time or throughput critical at peak times. No special design required and processing deadline is the next business day
   3. Response time and throughput are critical during all business hours. Although no special design for CPU utilization is required, the process ing deadline requirements with interfacing systems pose constraints
   4. Stated user performance requirements are stringent and require a per formance analysis in the design phase
   5. Performance analysis tools needed in the design, development, and/or implementation phases to meet stated user performance requirements

4. *Heavily Used Configuration*—The volume of data and transactions placed on a particular hardware platform.

   Degree of Influence

   0. No operational restrictions
   1. Operational restrictions exist, but are not overly restrictive and no spe cial attention is needed
   2. Some security and timing considerations are needed
   3. Specific processor requirements for a specific component of the appli cation exist
   4. Stated operational restrictions exist and require special attention
   5. There are special constraints with respect to the distributed components of the system

5. *Transaction Rate*—Similar to GSC 3, the number of transactions handled by the application will be a performance consideration with respect to the design, development, implementation, and maintenance of the system.

   Degree of Influence

   0. No peak transaction period is anticipated
   1. A single peak transaction period (i.e., daily, weekly, monthly, etc.) is anticipated
   2. A peak transaction period will occur weekly
   3. A peak transaction period will occur daily
   4. Transaction rates are high enough that a performance analysis is required during the design phase
   5. Transaction rates are high enough to require performance analysis and, in addition, the use of performance analysis tools during the design, development, and/or implementation phases

6. *Online Data Entry*—The amount of data entered online will influence the design development, implementation, and maintenance of the applica tion. Note: these guidelines may not be realistic since they have not been updated to reflect most systems today.

   Degree of Influence

   0. All transactions are processed in batch mode
   1. 1—7 % of transactions are done interactively
   2. 8-15% of transactions are done interactively
   3. 16-23% of transactions are done interactively
   4. 24-30% of transactions are done interactively
   5. Over 30% of transactions are done interactively

7. *End User Efficiency*—The functions provided by the application may emphasize user efficiency. This may include

   Navigational aids
   Menus
   Online help/documentation
   Automated cursor movement

Scrolling Remote printing

Preassigned function keys

Submission of batch jobs from online transactions Cursor selection of screen data Heavy use of reverse video, highlighting, colors, etc. Hard copy user documentation of online transactions Mouse interface Pop up windows

As few screens as possible to accomplish a business function

Bilingual support Multilingual support *Degree of Influence*

0. None
1. 1-3
2. 4-5
3. Six or more but with no specific user requirements in terms of efficiency
4. Six or more and stated user requirements are strong enough to require design tasks for human factors to be included (e.g., minimize keystrokes)
5. Six or more and stated user requirements are strong enough to require special tools and processes to demonstrate that requirements have been achieved

8. *Online Update*—Related to the number of ILFs updated by the application.

   Degree of Influence

   0. None
   1. Online update of one to three files, but volume of updating is low and recovery is easy
   2. Online update of four or more files, but volume is low and recovery is easy
   3. Online update of major internal files internal logical files (ILF)
   4. In addition, protection from data loss is critical and must be specially designed and built into the system
   5. In addition, high volumes lead to high recovery cost considerations, whereby recovery procedures must be automated and cause minimal operator intervention

9. *Complex Processing*—Complex processing is a characteristic of the appli cation and includes:

   Sensitive control and/or application specific security processing

   Extensive logical processing Extensive mathematical processing

   A great deal of exception processing whereby incomplete transactions that may be caused by such things as TP interruption, missing data values, or failed edits must be processed again

* Complex processing to handle multiple input/output possibilities (e.g., multimedia or device dependence)

Degree of Influence

0. None
1. Any one
2. Any two
3. Any three
4. Any four
5. All five

10. *Reusability*—The degree to which the application will usable in other applications.

Degree of Influence

0. There is no reusable code
1. Reusable code is used within the application
2. Less than 10% of the application considers more than one user's needs
3. 10% or more of the application considered more than one user's needs
4. The application was specially developed to ease reuse. The application is customizable to the user at the source code level
5. The application was specifically designed to ease reuse. The applica tion is customizable to use at source code level by means of user parameter maintenance

11. *Installation Ease*—The ease or degree of difficulty during conversion and installation.

Degree of Influence

0. No special considerations stated by the user. No special setup required
1. No special considerations stated by the user. However, special setup required for installation
2. Conversion and installation requirements stated by the user. Conversion and installation guides provided and tested, but impact of conversion is not considered important
3. Conversion and installation requirements stated by the user. Conversion and installation guides provided and tested, but impact of conversion is considered important
4. In addition to 2., automated conversion and installation tools were pro vided and tested
5. In addition to 3., automated conversion tools were provided and tested

12. *Operational Ease*—The efficiency and effectiveness of startup, backup, and recovery procedures that were provided and tested during the system testing phase.

Degree of Influence

0. No special considerations were stated by the user other than normal backup procedures

1-4. Select the following items that apply to the application. Each item has a value of one unless noted otherwise:

Effective startup, backup, and recovery processes were provided, but operator intervention is required.

Effective startup, backup, and recovery processes were provided, but no operator intervention is required (count as 2 items).

The application minimizes the need for tape mounts. The

application minimizes the need for paper handling.

5. The application is designed for unattended operation—that is, no operator intervention is needed other than to start or shut down the application. Automatic error recovery is a feature of the application.

13. *Multiple Sites*—The degree to which the application has been designed specifically to be installed and operated at multiple sites and/or for multiple organizations.

Degree of Influence

0. Only one user/installation site is required
1. Needs of multiple sites were considered and the application is designed to operate only under identical hardware and software environments.
2. Needs of multiple sites were considered and the application is designed to operate only under similar hardware and software environments.
3. Needs of multiple sites were considered and the application is designed to operate only under different hardware and software environments.
4. Documentation and a support plan are provided and tested to support the application at multiple sites as described in 1. or 2.
5. Documentation and a support plan are provided and tested to support the application at multiple sites as described in 3.

14. *Facilitate Change*—The degree to which the application was developed to facilitate change.

Degree of Influence

0. No special user requirements were stated to minimize or facilitate change

1-5. Select the items that apply to the application:

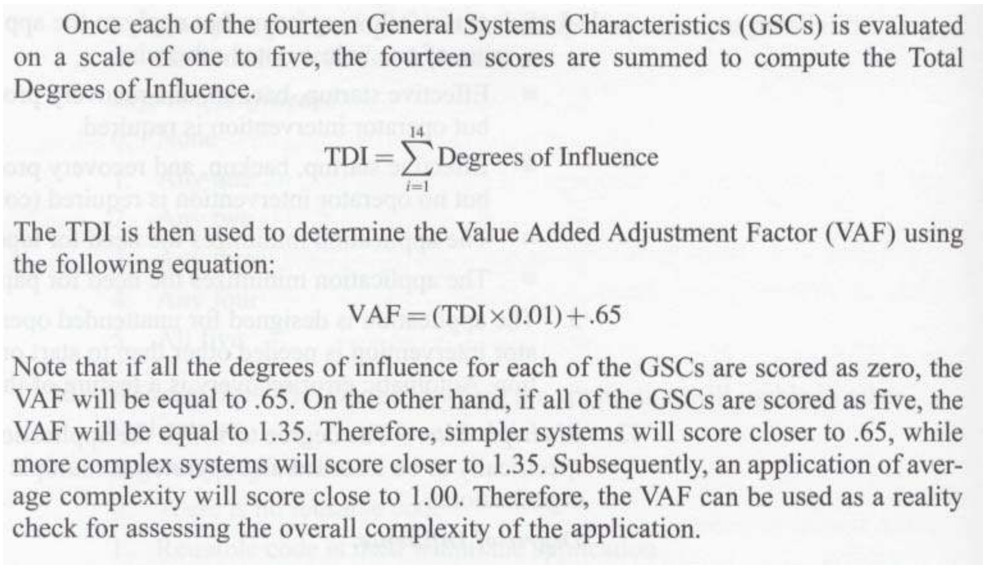
Flexible query/report facility is provided to handle simple requests—i.e., and/or logic is applied to only one ILF (count as 1)

Flexible query/report facility is provided that can handle requests of average complexity—i.e., and/or logic applied to more than one ILF (count as 2 items)

Flexible query/report facility is provide that can handle complex requests—i.e., and/or logic combinations on one or more ILFs (count as 3 items)

Control data is kept in tables and maintained by the user online. Changes take effect next business day

« Control data kept in tables and maintained by the user online. Changes take effect immediately (count as 2 items)



Once each of the fourteen General Systems Characteristics (GSCs) is evaluated on a scale of one to five, the fourteen scores are summed to compute the Total Degrees of Influence.

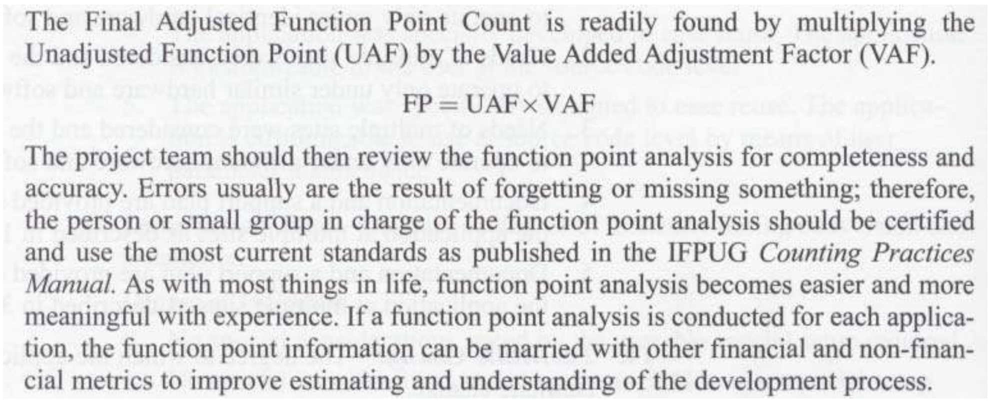
$$\mathrm{TDI} = \sum_{i=1}^{14} \text{Degrees of Influence}$$

The TDI is then used to determine the Value Added Adjustment Factor (VAF) using the following equation:

$$\mathrm{VAF} = (\mathrm{TDI} \times 0.01) + .65$$

Note that if all the degrees of influence for each of the GSCs are scored as zero, the VAF will be equal to .65. On the other hand, if all of the GSCs are scored as five, the VAF will be equal to 1.35. Therefore, simpler systems will score closer to .65, while more complex systems will score closer to 1.35. Subsequently, an application of average complexity will score close to 1.00. Therefore, the VAF can be used as a reality check for assessing the overall complexity of the application.

Step 7: Calculate the Final Adjusted Function Point Count



The Final Adjusted Function Point Count is readily found by multiplying the Unadjusted Function Point (UAF) by the Value Added Adjustment Factor (VAF).

$$\mathrm{FP} = \mathrm{UAF} \times \mathrm{VAF}$$

The project team should then review the function point analysis for completeness and accuracy. Errors usually are the result of forgetting or missing something; therefore, the person or small group in charge of the function point analysis should be certified and use the most current standards as published in the IFPUG *Counting Practices Manual.* As with most things in life, function point analysis becomes easier and more meaningful with experience. If a function point analysis is conducted for each application, the function point information can be married with other financial and non-financial metrics to improve estimating and understanding of the development process.

BIBLIOGRAPHY

Albrecht, Allan J. 1979. Measuring Application Development Productivity. Proceedings SHARE/GUIDE IBM Applications Development Symposium, Monterey, Calif., Oct 14-17, 1979.

Albrecht, A. J. and J. E. Gaffney. 1983. Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation. *IEEE Transactions Software Engineering,* SE-9(6): 639-647.

Boehm, B. W. 1981. *Software Engineering Economics.* Englewood Cliffs, N.J.: Prentice Hall.

Dekker, C. A. 1999. Managing (the size of) your projects: A project management look at function points. *Crosstalk: The Journal of Defense Software Engineering,* February: 24—26.

Dennis, A. and W. B. Haley. 2000. *Systems Analysis and Design: An Applied Approach.* New York: John Wiley. Garmus, D. and D.
Herron. 1996. *Measuring the Software Process.*
Upper Saddle River, N.J.: Prentice Hall.

Jones, T. C. 1998. *Estimating Software Costs.* New York:
McGraw-Hill. Longstreet, David. *Function Point Training and Analysis Manual.*
Longstreet Consulting Inc, Revision Dates: Feb. 2001, 30 Aug. 2001, 1 March 2002. <http://www.SoftwareMetrics.Com/free-manual.htm>. McConnell, S. 1996. *Rapid Development: Taming Wild Software*
Schedules. Redmond, Wash.: Microsoft Press.