IT METRICS STRATEGIES
Helping Management Measure Software and Processes and their Business Value

Eight Commonly Asked Questions About Size Metrics

by Michael Mah

When I conduct a presentation on software measurement, no subject generates more debate than project size. I often describe the Software Engineering Institute's (SEI) core measures of size, time, effort, and defects and find that members of the audience appreciate what the last three measures are but need to know more about that elusive first metric, size. Below are eight commonly asked questions on the subject along with responses that might help clarify matters.

1. "I usually think of size as the number of hours spent on an IT project. What do you mean by size?"

A common mistake. It's not unusual for a manager to ask how big a project is and get a response such as "13,840 hours."

Here's where the mess starts. In this case, the real question was "How much functionality was produced by the team?" But the project leader answered with a tally of the work effort expended and made things worse by using the wrong unit — hours instead of person-hours. Hours is a unit of elapsed time, as in, "It took me more than two hours to drive to Boston." No wonder everyone's confused: people aren't speaking the same language in this exchange.

Continued on page 2.

Function Points: When Are They Appropriate?

by David Garmus

The use of software metrics to manage and control project development and delivery is an accepted industry-wide best practice. Even though standardized metric definitions and practices are not yet fully developed, measurement programs are evident in more than 80% of IT organizations today. However, the content and the deployment of these measurement programs vary widely across IT organizations. No IT manager will argue with the adage you can't manage what you don't measure. What is often debated is the effectiveness and usefulness of particular metrics. Among those measures often disputed are function points.

Arguments for and against the use of function points abound. Everyone agrees there is a need for an accurate and reliable sizing

Continued on page 4.

march 2001 vol. VII, no. 3

executive summary

Size matters. This issue of ***ITMS*** revisits size metrics, which many IT organizations find perplexing. In an attempt to clear up the confusion, I've written "Eight Commonly Asked Questions About Size Metrics."

Chances are, you've heard some of these questions echo in the halls of your IT organization — and for good reason. Size metrics can be very controversial. Whenever the subject arises, it's not unusual for camps to form and for an almost religious-like fervor to engulf the debate. Many fall into the trap of arguing over which metric is "good" and which is "bad." What's better: function points or source lines of code? I'm right! You're wrong! But what about when both sides are right? That's tricky because, indeed, the world is not black and white. It's the shades of gray that complicate things. This article is intended to help you discuss the gray areas.

The second article in this issue is from someone new to ***ITMS***, but a seasoned veteran in the field — David Garmus of the David Consulting Group. Garmus is also the current president of the International Function Point Users Group. His article is entitled "Function Points: When Are They Appropriate?" Many IT folks are eager for answers to that question, and Garmus provides an excellent treatment of the subject for organizations considering the appropriateness of function points.

Finally, Carol Dekkers — always an excellent source of knowledge in this arena — shares an article entitled "Demystifying Function Points: Clarifying Common Terminology." This piece is invaluable for those who get dizzy from the jargon that hangs around size metrics. (For example, what is a "user"?) Language often leads us into situations people don't seem to understand each other. Dekkers helps us out with a crisp explanation of the language of function points.

Regardless of whether you use the function point metric, these articles will give you a headstart in solving those nagging size questions.

Michael Mah, Editor

CUTTER CONSORTIUM

Continued from page 1.

Think of size as what a team produced for a project that was completed. (We'll talk about in-progress and new projects later in this issue.) You're counting what was built by the team. That team might have comprised 10 people working over a period of 8 months. If what was built involved software of say, 100,000 source lines of code comprising 1,000 function points, then that's size.

Now, for the time. They took 8 months from start to finish. They expended 80 person-months of effort, which is 8 person-months per month times 10. If the team worked a standard 40-hour work week with 173 person-hours in an average month (4.33 weeks), that's 173 times 80, or 13,840 person-hours.

2. "Do we have to use function points or lines of code? If not these, then what?"

Function points work for some people, and this method can be a good fit if your applications involve "create, read, update, delete" against an underlying database. Code metrics are also valuable. It doesn't have to be either/or. I think polarizing things as either black or white is a mistake.

Ask yourself, "How do I view what we produce?" One group I asked replied, "Business processes [simple, moderate, and complex], which is how our end users view IT applications." Another said, "We quantify our systems in terms of the number of modules." Another replied, "C++ objects."

I find that people are more at ease thinking in the language of the world they live in. Once they are comfortable with that, you can go from there. Later, if you can examine how large your modules are in terms of code or function points, fine. You can go to another layer of detail later.

3. "I'm confused about the different ways of viewing size. I don't understand how they're related or why I should bother."

It's understandable. If we separate size from effort, the idea is to quantify and understand the size of what we've built in the past. This helps us scope new projects by analogy. After you have an understanding of the scope in terms of size, you can estimate the effort.

When you're looking at a completed project, I like to use the analogy of an aquarium tank: You could say that it holds 100 gallons of water. Or you could express it in liters. There are 128 fluid ounces in a gallon, so you could also express it as 12,800 fluid ounces. Any or all are valid.

The same goes with software size. You might initially express it in objects, modules, or programs. You can also go a step further and express the number of function points and/or the code that makes up these objects, modules, or programs. All are valid.

Once you understand this, several things open up to you. You can benchmark your IT productivity. Outsourcers or companies looking to outsource find this critical. You can also use any of the excellent commercial estimating models to forecast time, effort, and, with some of the more sophisticated ones, the expected quality and reliability. Forecasting time and effort manually is difficult. It takes time, and things change so fast you can't keep up. Computer models make this much easier.

4. "What are the reasons why many people don't size their projects?"

Many folks seem worried about being judged or evaluated based on their "output" and feel that size metrics won't reflect the hard work they do. Employees don't always express that, but you can feel it. Fear can be a significant factor, but "real" men (and women) usually don't like to admit they're scared — it clashes with how they view their identity.

Editorial Office: Clocktower Business Park, 75 South Church Street, Suite 600, Pittsfield, MA 01201, USA. Tel: +1 413 499 0988; Fax: +1 413 447 7322; E-mail: mmah@cutter.com.
Circulation Office: ***IT Metrics Strategies®*** is published 12 times a year by Cutter Information Corp., 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA. For information, contact: Tel: +1 781 641 9876 or, within North America, +1 800 492 1650, Fax: +1 781 648 1950 or, within North America, + 1 800 888 1816, E-mail: service@cutter.com, Web site: www.cutter.com/consortium/.
Editor: Michael Mah. Publisher: Karen Fine Coburn. Group Publisher: Kara Lovering, Tel: +1 781 641 5126, E-mail: klovering@cutter.com. Production Editor: Lori Goldstein, +1 781 641 5112. Subscriptions: $485 per year; $545 outside North America.

5. "I can count code, but some parts of a project might not result in a lot of code, while others do, for the same amount of work-effort. What can I do about this?"

Separate size from complexity. (Most people combine these two.) Tough stuff takes more work-effort; easier stuff takes less. First get the raw numbers on the work output, then address the difficulty factor as a separate issue.

6. "Many of our applications involve multiple languages. How do I handle this?"

This is the norm, not the exception. Keep profiles in each domain and draw a pie chart. As you begin to understand how the pieces of the pie fit together, you'll acquire a new understanding on how projects of different mixes behave. This can give you valuable information when planning new projects.

7. "What does sizing do for me at the end of a project? At the beginning? During?"

Great question. In the first case, you're counting something that physically exists, even if it's this nebulous thing known as software. Your source code libraries hold the "stuff" that runs on the hardware that plays the music, whether that's a mainframe, a mini, a workstation/PC, or a black box sitting in an airplane's cockpit. You can count it. Follow the Carnegie Mellon SEI guidelines for logical source instructions (known as "code"). Yes, we are talking about software in many cases.

If you have up-to-date documents on the application, you can count the function points if the architecture definitions fit. Oh, you don't have up-to-date documentation? Well, you're not alone.

For a project just starting out, nothing exists yet to count. You're estimating the amount of functionality. Or you're counting what people are asking of you, such as 48 work requests, which might later amount to a certain number of programs, modules, function points, and ultimately — and I hate to say it — code. (Yes, we are still talking about software in some cases.)

In the middle of a project, you're between two worlds. If you're 4 months into a 10-month project, you're done with some parts and not done with others. You want to quantify the amount you've produced so far and estimate the amount that you have left. Maybe you've written and modified the programs needed to accomplish 18 of the 48 work requests as part of Increment 1, and you've got 30 left to go. Then you have to integrate the whole thing and calculate how long the alpha and beta tests will take.

The key is to think of this in terms of volume of functionality. Most people only think of midstream progress as having expended a certain amount of person-hours (and the associated budget) to date. That's just looking at how much you've spent, not what you've produced for your money.

8. "What about Web, object-oriented development, or multitier client-server? Can function points work here?"

Some people feel this is like trying to fit a round peg in a square hole; others don't see it this way. It depends on your situation. However, at the time that function-providing elements were defined in the early 1980s, the world was quite different. Most IT computing was IBM back-office mainframe stuff, with lots of batch processing.

Today, IT has gotten incredibly complex, with technology pressing into domains that some never dreamed would exist. There are highly sophisticated communications including wireless, near-time processing, embedded elements, Internet protocols, and the like. As Dorothy said in *The Wizard of Oz*, we're not in Kansas anymore, Toto. Trying to force function points into feature points, object points, etc., seemed futile to me and never took hold. (There's that old saying that if you have a hammer, the whole world can look like a nail.)

I think the answer is to find out what works in your world. Function points can be an excellent metric to get people thinking in analytical ways that can be productive. Many architectures can benefit from being sized that way. Others need a different technique.

Try not to completely discount any sizing approach. As you shape your understanding of your IT applications from various perspectives, you'll gain knowledge to help you better plan and manage IT projects in this high-pressure world. Some people moan

about what this might take. But if you think education is expensive, try ignorance.

Reference

Dekkers, Carol, and Ian Gunter. "Using Backfiring to Accurately Size Software: More Wishful Thinking Than Science?" ***IT Metrics Strategies***, Cutter Information Corp., November 2000.

Mah, Michael C. "Sizing Up Your Promises and Expectations." ***IT Metrics Strategies***, Cutter Information Corp., September 2000.

Mah, Michael, C. "Determining Your Own Function Point and Lines of Code Proportions — Three Things to Watch Out For." ***IT Metrics Strategies***, Cutter Information Corp., November 2000.

Function Points: When Are They Appropriate?

Continued from page 1.

metric, and many feel that function points are the best sizing metric available. The issue that is most often questioned is whether function points are practical and applicable in all situations. This article will address where function points work, where they may not work, and some alternative approaches.

An Overview of Function Points

Function points are a synthetic sizing or normalizing method, much the same as square feet or cubic yards, that permit the calculation of a relative size for individual software projects, applications, or subsystems, even in their early requirements stages. They represent a unit of work. Function point analysis typically occurs when a developer wants to size and estimate development time and effort for an application or a project. The analysis represents a business user's view that can be used as a vehicle to communicate what in fact will be delivered as a product. Function points include data and transactional functionality.

Logical user data groups maintained within the application are identified. External inputs add, populate, revise, update, or change the data stored. The user data groups are ultimately processed within an application to permit the extraction of data and the generation of external outputs or external inquiries. These data groups are classified as internal logical files (ILF). Their complexities of low, average, and high are based upon their record element types and fields.

Data is often extracted from ILFs belonging to other applications. If a data group is maintained within a separate application, which is read or referenced to assist in processing external inputs, external outputs, or external inquiries, that data group is classified as an external interface file (EIF). As with ILFs, EIF complexities are based on the record element types and fields used.

Data received from outside the application boundary that provides control functions or maintains (adds, changes, or deletes data) in an ILF are identified as external inputs (EIs). ILFs and EIFs are referenced and fields are counted to determine the complexity of EIs.

Data generated within the application that exits the boundary is counted as an external output (EO) or an external inquiry (EQ). Data output (printed information, screen display, or a file sent to other applications) that is retrieved from any combination of ILFs or EIFs is counted as an EQ as long as it does not contain derived or calculated information, provide control functionality, or maintain (add, change, or delete data in) an ILF. If a data output from an application contains derived or calculated information, provides control functions, or maintains an ILF, it is counted as an EO. ILFs and EIFs are referenced and fields are counted to determine the complexity of EOs and EQs.

Low, average, and high ILFs, EIFs, EIs, EOs, and EQs are totaled using International Function Point Users Group (IFPUG — www.ifpug.org) matrices to determine the total unadjusted function points. There are 14 general system characteristics (GSC) evaluated based on their degree of influence on the application. GSCs influence the final calculation of the adjusted function point count.

Critical to the ongoing success of function points as a viable software metric is the work that is accomplished by IFPUG. Since 1986, IFPUG has consistently grown in membership and in importance to the software measurement community. Today, IFPUG's membership includes thousands of individual, corporate, educational, and institutional members from more than 30 countries.

IFPUG is a not-for-profit, member-run user group. It's mission is to be a recognized leader in promoting and encouraging the effective management of application software development and maintenance activities through the use of function point analysis and other software measurement techniques.

Inherent Value of Function Points

Function point analysis is an accepted international standard for the measurement of software size. Function points are well defined in IFPUG's *Function Point Counting Practices Manual*, and their definitions, descriptions, and rules are maintained by that organization. The guidelines are frequently updated to provide consistency and to recognize new technologies. IFPUG serves to facilitate the exchange of knowledge and ideas for improved software measurement techniques. Function points are most effective when used with industry data points to achieve accurate estimating.

Because of their popularity, function points have become the basis for many industry comparisons and benchmarks; for example, best-in-class or industry-average delivery rates. Function points are a critical input to most of the industry-accepted estimation tools. Development project function point size is considered with nonfunction-point-related application characteristics (e.g., application type, performance requirements, security, algorithmic content, and platform) and project attributes (e.g., skill levels of the developers, development languages to be used, potential reuse opportunities, methodologies and technologies to be applied, and tasks to be performed) to estimate cost and resource requirements. An effective estimating model considers three elements: size, complexity, and risk factors. When factored together, they result in an accurate estimate, as displayed in Figure 1.

Function point analysis can be introduced early in the estimation process and reevaluated whenever there is a change of scope or as a new phase of the development process commences. Because function points represent the functionality that a business user requests (whether that user is a marketing person, business analyst, factory manager, banker, or buyer), it is never too early to accomplish a functional analysis. In fact, gathering the interested parties together early in the software project proposal phase to achieve a functional analysis will ultimately make the developers' job easier and will ensure that the users state their case for what they want developed. Of course, this does not automatically ensure the accuracy of the count or estimate, but it does permit better definition of the project requirements.

The timing of function point counts will vary based on the particular status of an application. Less information is available early in the development process, and the count and any resulting estimate will be less accurate. Significantly more information is available as an application is developed and delivered. Early in the process, the only information available might be verbal. During the devel-

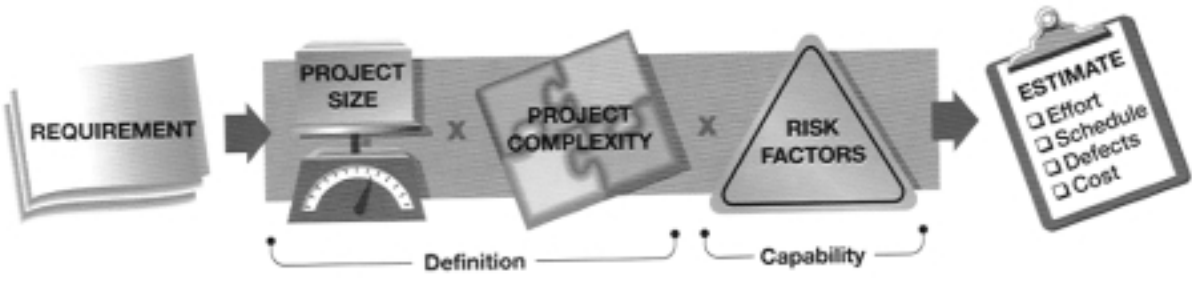

Figure 1 — The estimating principle.

opment process, the information available to assist in counting increases to include some of the following documents, any of which would be helpful in determining function point counts:

- Project proposals
- High-level system diagrams (indicating relationships to other interacting applications)
- Entity relationship diagrams
- Logical data models
- Data flow diagrams
- Object models
- Process models
- Requirement documents
- Prototypes
- Functional specifications
- Use cases
- System specifications
- Detailed design specifications
- Physical design models
- Operational models
- Program and module specifications
- Feature cases
- File layouts
- Database layouts
- Screens and screen prints for online systems
- Copies of reports or report layouts
- Test cases (features)
- User manuals and other technical documentation
- Training materials
- System help

Occasionally, insufficient documentation or no documentation exists, and the counter must rely on the knowledge of one or more business or systems experts to describe the application.

Types of Function Point Counts

The first step in the function point counting process is the determination of the type of function point count to be conducted. There are different purposes for performing function point counts. Development projects (including new development and enhancements to existing applications or packages) are sized to assist in estimating or measuring project costs. An application is sized to determine the current value as well as to estimate or measure maintenance support cost or maybe to estimate replacement cost. The three possible types of function point counts are described below.

Development project function point counts measure the functionality provided to end users with the first installation of the application. They include the functionality that is counted in the initial application function point count. A development function point count must often be updated as the development process proceeds. Subsequent counts would not be "start from scratch" counts, but they would validate previously identified functionality and attempt to capture additional functionality, commonly called scope creep. Figure 2 displays the various lifecycles of development through implementation and indicates periods during which counts could occur.

Enhancement project function point counts measure modifications to existing applications and include the combined functionality provided to users by adding new functions, deleting old functions, and changing existing functions. After an enhancement, the relevant application function point count must be revised to reflect any changes in the application's functionality.

Application function point counts measure an installed application. They are also referred to as baseline or installed counts, and they recognize the current functionality provided to end users by the application. An activity's total installed application function point count represents the sum of the application counts for all installed applications that are being used and maintained.

Project function point counts correlate to project development effort. Function points

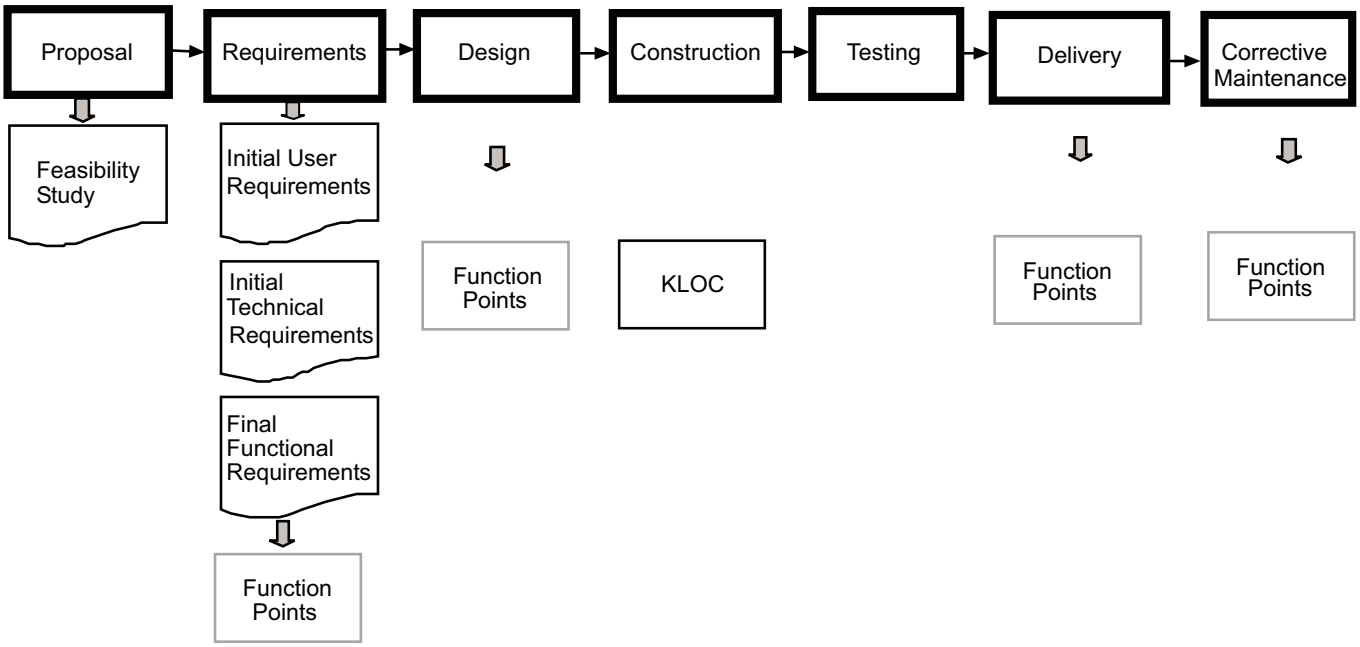

Figure 2 — The development lifecycle.

are a key metric in estimating accurately or in measuring what was delivered correctly. This is especially true for large projects. Effort on smaller projects with less than 50 function points or on projects that relate to tool/language/database updates, performance improvements, Y2K fixes, or other bug fixes are not sized well with function points; even if a count can be reached, the count does not correlate with effort.

Application function point counts are typically used separately from size and estimate requirements for application maintenance. Less accuracy is necessary on each application count when the number is intended to reflect a total baseline size; consequently, some creative estimating alternatives are possible.

What and When to Size

The David Consulting Group (www.davidconsultinggroup.com) conducted a detailed study in 1999 through the support and participation of several client companies. The focus of the study was to determine the cost and accuracy of various counting techniques, such as IFPUG function point counting, approximation, estimation, and backfiring, as defined below.

IFPUG — a complete and detailed count typically performed on highly visible systems, systems that are core to the business, or systems that have been the subject of frequent change requests

IFPUG Limited — accuracy is a primary concern, but average weightings are applied

Robust approximation — the best of the approximation methods that can be used when accuracy is not of primary concern but full functionality needs to be recognized

Ratio — typically used in instances where all data can be identified and logically parsed into user-identifiable groupings

Expert — used in sizing commercial off-the-shelf packages or with common applications where the analyst is familiar with similar types of applications

Delphi — the least effective of the approximating and estimating techniques; used when an organization's portfolio already has a certain percentage of IFPUG or IFPUG limited counts available

Backfire calibration — a backfire method that utilizes a customized backfire value; used when accuracy is not an issue, but a sense of overall functionality size being supported is necessary

Backfire calculation — same as calibration, except industry backfire values are used

The expected accuracy and effort required for the above sizing methods is displayed in Figure 3.

It is not always practical or necessary to invest in the full counting of each application in an organization's portfolio. Alternative approaches to IFPUG function point counting should be considered when it may not be appropriate to invest the effort required to conduct an accurate function point count.

Individuals frequently raise roadblocks to counting, such as the following:

- It is too early.
- There is not enough information available.
- There is not enough time to count.
- We have no expertise.
- This project is different.

Many of these objections can be overcome by using some of the previously discussed alternatives; approximation is possible early in the project; it takes very little time to do a quick count; and counting can be outsourced when local counters are not available.

Alternative Sizing Techniques

Many alternatives exist, but none has experienced the widespread acceptance of function points. Mark II and feature point methods come close to function points: they can be counted at the same time, with the same level of detail, and with the same precision as function points. Bang and 3D require a greater degree of detail to determine size and consequently make earlier counting more difficult. Early estimates of source lines of code (SLOC) are very inaccurate for average to large projects, and coding is such a limited aspect (perhaps 5%-15% of the total project effort) for software development projects that using SLOC for estimation at any time tends to be erroneous. Most of the recently introduced source code languages are not conducive to counting with SLOC.

Descriptions of some of these sizing and estimating metrics follow.

Mark II Method

The Mark II method, introduced by Charles Symons, has been used almost exclusively in the UK. Mark II uses the same basic parameters as function points in its calculations. However, it makes use of fewer parameters and was intended to:

- Reduce the subjectivity in dealing with files by measuring the number of entities

Count Type	Accuracy	Cost
IFPUG	+/- 5%	1-3 days
IFPUG Limited	+/- 25%	1-3 days
Approximation	+/- 35%	½ day
Ratio	+/- 50%	<½ day
Expert	+/- 50%	<½ day
Delphi	+/- 100%	<¼ day
Backfire	+/- 100%-400%*	varies

* Variation occurs based upon language levels

Note: The cost is based on an average size application (250-1200 function points) and will vary for applications outside of that size range.

Figure 3 — Accuracy and cost of sizing methods.

and their performance as they move through the data structure

- Modify the function point method to compute the same numeric totals regardless of application boundary as a single system or as a set of related subsystems
- Focus on the effort required to produce the functionality rather than on the value of the functionality delivered to the users
- Add six complexity factors to the IFPUG general system characteristics

Feature Points

The feature point method, developed by software development expert Capers Jones, is a superset of the function point method. It includes an additional component — algorithms — adding to the set of the five function point components: EIs, EOs, EQs, EIFs, and ILFs. With the other function point components, the algorithm component is assigned a weighted value. When using the feature point method, the values assigned to ILFs are reduced. Jones has stated publicly that feature points are no longer supported and that the use of IFPUG function points are preferred.

3D Function Points

3D function points were publicly introduced by the Boeing Computer Services Software Metrics Team in 1991. They were developed in response to the call for a technology-independent size metric suitable for all domains.

The 3D method is based on the premise that the application problem can be expressed in three dimensions: data, function, and control. Each dimension contains some of the characteristics that create complexity in a problem. Sometimes one dimension dominates, but all dimensions of the problem must be analyzed if accurate measurement is desired.

The 3D method identifies characteristics from each dimension that can be measured directly. Data-strong problems are typically associated with IS/business software environments. The data dimension characteristics are taken directly from the function point guidelines and include evaluation of EIs, EOs, EQs, ILFs, and EIFs. Function-strong problems are associated with scientific/engineering environments. The characteristics of the functional dimension include the number and the complexity of functions that represent internal processing required to transform input data into output data, as well as the sets of semantic statements that govern the process. Control-strong problems are associated with real-time environments. The characteristics of the control dimension include system states and transitions.

Bang Metric

The Bang metric, published by Cutter Technology Council Fellow Tom DeMarco, is similar to the function point metric, but Bang (like 3D) requires information on transitions and states and separates data-strong applications from function-strong applications. The Bang metric is deemed more appropriate for real-time, telecommunications, and scientific software projects than IS projects. However, Bang does not have the number of users nor the documented successes, even in those environments, as function points.

The Bang metric considers functional and modified functional primitives; input, output, and stored data elements; states and transitions in a state transition model; entities and relationships between entities; data relationships; and data tokens.

COCOMO

The Constructive Cost Model (COCOMO), introduced by Barry Boehm, evaluates cost factors to estimate effort-months. This model requires the number of source lines of code to be delivered as its major input. Since COCOMO was released to the public domain, many models and variations have appeared in the marketplace. The latest release of COCOMO permits the use of function points as well as object points (which are determined by applying weights to screens, reports, etc.).

Cost factors are evaluated and weighted within COCOMO for application complexity and software reliability; execution, memory, and environmental constraints; development personnel skill levels; tools and technologies; and a variety of other capabilities.

COSMIC

As recently as 1998, the Common Software Metrics International Consortium (COSMIC— www.cosmicon.com) introduced COSMIC full function point as a new measurement alternative for measuring the functional size of software. The basic principles of the method were established, drawing on the best features of the existing function point methods, the International Organization for Standardization standard for functional sizing measurement, and new ideas. The first formal definition of the method was released in October 1999. It is too early to tell whether or not this method, still in its infancy, will provide any additional benefit to the software community at large.

If Not Function Points, What?

Just like in the recent US presidential election, parties typically understand only their views in the use of function points — either strongly in favor or strongly against. However, any method can and maybe should be used concurrently with other sizing and estimating methods. If you are using an alternative sizing method, there can be no harm in attempting the use of function points on a trial basis. Determine how the two methods correlate, and perhaps you will see some added value in looking at the project with a different point of view. Function point analysis will at least provide a list of processes that must be incorporated into the deliverable.

Function points are well defined, supported by IFPUG, supplemented by a large number of industry data sources and benchmarks, and required as an input to many estimation tools. The alternative approaches to counting discussed in this article may provide the answer to some of your objections.

Before you dismiss the function point methodology entirely, it is recommended that you give it — and the alternative approaches — a serious trial, particularly for large projects.

About the Author

David Garmus is an acknowledged authority on the sizing, measurement, and estimation of software application development and maintenance. He has more than 25 years of experience in managing, developing, and maintaining computer software systems. He has served as a university instructor, teaching courses in computer programming, system development, information systems management, data processing, accounting, finance, and banking. He received his bachelor's degree from University of California, Los Angeles, and an MBA from Harvard University. Mr. Garmus is president of the International Function Point Users Group (IFPUG) and a member of the Counting Practices Committee. He previously served IFPUG as chair of the Certification Committee, chair of the New Environments Committee, and on the board of directors as director of applied programs and vice president. He is a member of the IEEE Computer Society and a member of its Standards Association. Mr. Garmus and Cutter Consortium Senior Consultant David Herron are the principals and founders of The David Consulting Group (www.davidconsultinggroup.com). They coauthored *Measuring the Software Process: A Practical Guide To Functional Measurement* and *Function Point Analysis: Measurement Practices for Successful Software Projects*.

Mr. Garmus is a recognized author and lecturer who has addressed audiences worldwide on functional measures, software process improvement, and estimation. He has also contributed numerous articles to such periodicals as *Software Development, Data Manager, Software Testing & Quality Engineering, Methods & Tools, Data Manager, and CrossTalk.* His email address is dcg_dg@compuserve.com.

Demystifying Function Points: Clarifying Common Terminology

by Carol A. Dekkers

A challenge in implementing function point analysis (FPA) is making it understandable to developers, cost analysts, and customers alike. Because function points are based on functional user requirements (what the software does), irrespective of the physical implementation (how the software is implemented), users of the method must think in terms of the logical functional requirements. This can be especially difficult for developers who spend their days focused on providing physical software solutions (involving how the software will be built).

However, just as an architect begins by drawing a floor plan to meet the owners' needs, software project managers and developers begin by documenting the functional user requirements articulated by their customers. FPA examines these logical (also called functional) user requirements to determine the functional size of the software.

The difficulty that sometimes arises with developers is twofold:

- A developer's job concentrates on the physical aspects of designing and implementing software, similar to how a master plumber or master electrician concentrates on the physical aspects of building a house.
- The FPA method uses terms that are well known in the IT industry, but the meanings of particular words are different.

This article focuses on several keywords that are the most problematic when introducing function points. By simply making your developers aware of these differences in meaning, you can achieve higher levels of measurement success and less resistance in FPA implementation.

All of the functions that are evaluated in function point counting are logical user functions, that is, they are part of the logical user requirements for the software. Readers who want further details about function points are encouraged to obtain the *Function Point Counting Practices Manual* (version 4.1) from the International Function Point Users Group (IFPUG).

Terms That Cause Confusion

The following terms are used within both the FPA method and IT. The confusion arises because their general use in IT often conveys a different meaning from that used in function point counting:

- Logical
- User
- Application (system)
- Project
- File
- Enhancement

Each term (and the confusion it causes) is explained in the sections that follow.

Logical

In IT, the word "logical" is usually associated with logical database layouts or logical data models. However, in many situations, some physical considerations creep in to even the most logical of data models.

When the term logical is used in function point counting, it refers to the conceptual or functional user requirements and excludes physical implementation or design requirements. Logical user requirements are those that an experienced user in the subject matter area would identify as requirements of the software. Logical or functional user requirements describe what the software must do, and they do not include how the software will do the functions. Function points measure the size of these functional user requirements only. Design and quality considerations, although important to the software construction, are not part of the

logical size of the software; therefore, they are not counted in function points.

This is similar to the size of a house being measured as the number of square feet or square meters contained within a floor plan; the number stays the same no matter how the house is constructed. In functional size measurement, the function point count reflects only what the application must do, not how it will do it.

Confusion can arise when the word logical is used in conjunction with words that sound physical, like screen or report. A logical screen may consist of one or more connected physical data entry screens that support a single function. Everything in function points is counted from a logical user viewpoint, and novice counters need to think "logically" when they are counting function points.

User

The term "user," as typically used in IT, refers to a living, breathing person who interacts with or uses the software. This is the most frequent answer given when one asks a developer, "What is a user?"

The terminology problem with the word user arises because the function point use of the word has a wider meaning. The IFPUG *Function Point Counting Practices Manual* defines user as "any person that specifies Functional User Requirements and/or any person or thing that communicates or interacts with the software at any time." [1]

In other words, for function point counting, users can be people, applications, departments, or other external parties — in short, anything that requires data from, or provides data to, the software. Functional user requirements include the logical business processes of many users. Users can include other software applications, physical persons, external government bodies, departments, animals (if they trigger a process in the application, such as in security systems), things (such as pressure in a pipeline system) — anything that interacts by sending or receiving data across the boundary of the software application.

This difference in meaning can cause some countable functions to be overlooked by developers because they don't appear to be requirements provided for or by a human user.

Application (System)

The terms "application" and "system" in data processing are often used interchangeably and are usually tied to the physical segmentation of the software. Here are some examples of how applications or systems may be broken down by developers:

- **Based on functions performed in batch or online — s**ometimes, each mode of physical implementation of a single set of cooperative functions may be split into separate "systems" by developers; for example, the batch accounts receivable system and the online accounts receivable system.
- **Based on the physical platform on which a subset of functions (or subfunctions) resides** — for example, the mainframe payment system and the PC payments system.
- **Based on the physical package(s) that make up a set of functions** — for example, the Access database application, the InfoMaker reports application, and the data entry application. There are many other derivatives.

The term application within the context of function point counting is defined in the *Function Point Counting Practices Manual* as "a cohesive collection of automated procedures and data supporting a business objective. It consists of one or more components, modules, or subsystems. Frequently used synonymously with System, Application System, and Information System." [2]

This means that an "application" in function point terms is a collective grouping of related user functions regardless of the platform, mode of operation, and physical IT subdivision of functions. In the examples above, the batch and online "systems" would be one "application" for function point counting. The second example would result in one payments "application" from the user perspective (which happens to be physically implemented across multiple platforms). The third example would likely also be a single application with various packaged

components used as part of how the software was delivered.

This difference in definition and usage of the word application can result in overcounting (e.g., if an application was function point counted as two applications [as an online count and a batch count, rather than properly as a single application], or undercounting (e.g., if all of the applications on a single hardware box were counted as a single application).

It is important to remember that in function point counting, an application represents how a user would logically view the system, while in IT, an application often represents how the system is physically implemented.

Project

The term "project" often differs between its IT and function point meaning. When used in systems development, project can take on a variety of meanings, even within the same organization. The word project can be used to describe variously:

- The scope of work that includes enhancement or development of several discrete software applications
- The scope of work including fixes/ maintenance of existing functions plus enhancement to other functions of a single software application
- Repairs of operating software upgrades of existing software
- Combinations of any of the above

In function point counting, the word project refers to the work product associated with the development or enhancement of a single application (system). The definitions in the *Function Point Counting Practices Manual* attest to this:

Project — a collection of work tasks with a time frame and a work product to be delivered. [3]

Development project function point count — a count that measures the functions provided to the users with the first installation of the software delivered when the project is complete. [4]

What this means to function point counters and developers is that a project in business or IT terms may equate to multiple function point projects and, therefore, translates into needing multiple function point counts, one per application involved. For example, if an IT project includes the development of a new hospital billing system plus enhancements to an existing hospital admittance system, the size of the overall IT project would involve two function point project counts: a development function point count of the new hospital billing system and an enhancement function point count of the changes to the hospital admitting system.

It is also worth noting that an enhancement project, whose total size encompasses the added, changed, and deleted functionality, will change the product or application size by the amount of functionality added less that deleted.

Once this difference in the term project is understood, it is simply a matter of communicating the function point counts in the right context. [5]

Enhancement

For business and IT professionals, the term "enhancement" or "enhancement project" refers to any project where the existing software is enhanced in terms of performance, appearance, function, operation, platform, usability, and so on. (Note that enhancement projects are often discerned by both users and developers from the term "maintenance," in that the latter pertains to work done to keep the existing software up and running.) The following are examples of what users and developers would refer to as software enhancements:

- Making changes/additions to hard-coded data within the system
- Populating new occurrences of dynamic data (e.g., having developers do data administration/table population as part of the project, either manually or with the use of tools)
- Upgrading software to be compatible with new database releases (such as a new Oracle release)
- Splitting one physical screen into multiple physical screens (but not changing the functionality)

- Changing error message text to be more user-friendly
- Cosmetic screen or report rearrangement, adding more screen highlighting, or creating additional menus (restructuring the navigation)
- Adding new peripheral devices (such as adding the ability to print to laser printers on a network that previously only allowed dot matrix)
- Testing one application to verify the expected effects of changes in another application
- Increasing the results list for an existing report by adjusting a filter to retrieve more valid values
- Adding data elements to an existing report

This is very different from enhancement in the context of function point counting, where it means functional enhancement. The *Function Point Counting Practices Manual* definition is "a count that measures the modifications to the existing application that add, change, or delete user functions delivered when the project is complete." [6]

Enhancement in function point terminology strictly refers to logical, functional enhancements, that is, only those modifications to the logical user requirements or elementary processes — external inputs (EI), external outputs (EO), external inquiries (EQ), internal logical files (ILF), or external interface files (EIF) — of the software. As such, if an enhancement project in business terms does not create new logical functions, modify existing logical functions, or remove logical functions from the software, then no function points can be counted. What this means is that out of the previous list of enhancements in the business and IT context, potentially only the last point (adding new data elements to an existing report) would count any function points. This is because the remaining list items are not considered to be functional enhancements, and therefore, they would score zero function points. [7]

This can be very frustrating to the uninitiated function point practitioner who does not have an appreciation for the importance of the context in which the term enhancement project is used.

File

The term "file," when used by system developers, usually evokes images of mainframe, transaction-oriented processing, and the term is used interchangeably with "dataset." Associated terms such as research files, output files, sort files, batch files, Excel files, and transaction files are still common vernacular today.

In function point counting, file is used to represent a logical grouping of data that is a requirement of users. The *Function Point Counting Practices Manual* defines file as "for data function types, a logically related group of data, not the physical implementation of those groups of data." [8]

The confusion emerges because ILF and EIF in function point terms refer to persistent data entities, not physical files or datasets. Here are some examples of physical files/datasets in IT terminology that would not be files (entities) in the context of function points:

- An input dataset could house transactions that will cause updates to master data in the application. (In FPA, this would count as one or more EIs because that is the logical user requirement. The physical housing of those processes happen to be in a dataset.)
- An output file could contain the electronic version of multiple, distinct reports or groups of data (e.g., US tax forms such as W2s and 1099s, etc. could all be housed on a single, physical output tape). In FPA, this would be counted as several EOs or EQs depending on the specific logical process involved (one for each of the unique functional user requirements). It does not change the functional user requirements whether there are several physical tapes or one physical tape — as long as the users receive the functionality.
- A physical restart file could contain incomplete, intermediate results and be used as input to an intermediate job step. This would be part of the physical implementation and would not be a logical

user requirement; therefore, it would not count.

- A table of user-maintained data on regions would be counted as an ILF if it were part of the user requirements and maintained by the application being counted.

The key is to remember that the word file in function points refers to a logical grouping of related data. This does not conform to a file or physical dataset in IT terms.

Summary

This list of commonly misunderstood words is not exhaustive, and other words and acronyms can form barriers to an

Table 1 — Summary of Terms

Term	Meaning in IT	Meaning in Function Point Counting
Logical	Typically includes both conceptual and design considerations. Even "logical" data models often contain physical components.	Refers to logical functions and logical user requirements. Conceptual, from a user business perspective. Does not include design or quality considerations. Reflects what the software must do, not how.
User	Physical person who uses or specifies requirements for the software.	Person, thing, other application, department, etc., that provides functional user requirements for the software.
Application (system)	Physical implementation of software. The boundary of an application or system often coincides with physical hardware or software boundaries.	A cohesive collection of automated procedures and data supporting a business objective.
Project	Depending on the organization can include new development, changes to one or more applications, or enhancements to multiple applications.	Pertains to the work product done on a single application: Development — the specification, construction, testing, and delivery of a new information system. Enhancement — the modification of an existing application.
Enhancement	Any modification to the software, including functional, nonfunctional, technical, cosmetic, data administration, or design changes that increase the business value of the software.	Functional modifications to the elementary processes of the application (e.g., new/ modified/removed functions: external inquiry, external outputs, external queries, internal logical file, or external interface file).
File	Dataset or physical assemblage of data, as in output file, input file, data file, research file, etc.	A logically related group of data, not the physical implementation of those groups of data.

understanding of function points. Table 1 (on previous page) summarizes the terms covered in this article. Functional size measurement and function points are not rocket science — they simply provide an objective, repeatable process for assessing the logical size of software based on functional user requirements. By understanding some of the terminology that can trip up developers and novice counters, this article will, hopefully, demystify the perceived complexities involved in function point counting.

References

1. *Function Point Counting Practices Manual,* version 4.1. International Function Point Users Group, 1999, glossary.

2. Ibid. Glossary

3. Ibid. Glossary

4. Ibid. Glossary

5. Note that support projects and maintenance projects (such as bug fixes, software upgrades, data changes, etc.) are not covered by the development and enhancement projects because the user functionality does not change. Function point support rates can be calculated to equitably allocate your human resources across applications. This ratio relies on the application base function point count rather than project counts. Contact the author via e-mail (dekkers@compuserve.com) for further information on how to manage maintenance initiatives using function points.

6. Ibid. Glossary

7. There is the possibility of a zero function point enhancement project (i.e., no functions were added, modified, or removed) where the performance of the application or other general system characteristics may have been modified, resulting in an increase or decrease to the adjusted function point size of the installed application. It is questionable whether an increase or decrease in the baseline for which the project did not alter any functionality could be considered as the enhancement project function point count.

8 Ibid. Glossary

About the Author

Carol A. Dekkers is the president of Quality Plus Technologies, Inc., a progressive management consulting firm specializing in making software measurement and function points meaningful to the IT industry. Ms. Dekkers is an acknowledged measurement expert, the host of the weekly *Quality Plus e-Talk* radio show, an author, a consultant, and an instructor. Ms. Dekkers is a past president of the IFPUG board of directors and currently serves in leadership positions for the Project Management Institute's Metrics Special Interest Group, the Quality Assurance Institute Conference Advisory Board, the American Society for Quality Software Division, and as project editor of the ISO Functional Size Measurement workgroup (ISO/IEC/JTC1/SC7 WG12). She has published more than 50 articles, of which a partial list and an article request form can be found at www.qualityplustech.com. She can be reached at dekkers@qualityplustech.com.