ti MÉTRICAS

A Empresa Líder em Métricas e Análise de Pontos de Função

Applying Function Point Analysis to Requirements Completeness

Carol Dekkers
Quality Plus Technologies Inc.
Mauricio Aguiar
Caixa Economica Federal

Requirements issues abound in system development despite many models and methods intended to verify that requirements are complete. This article highlights how function point analysis (FPA), the software sizing technique, delivers value as a structured requirements review. While its historical usage has been confined almost exclusively to quantifying software size, FPA is gaining popularity as a useful, structured method for reviewing requirements. When used during software development to verify requirements completeness, FPA delivers more than mere numbers for software size—the FPA documentation reflects the full, known set of functional user requirements.

There is a wide diversity of traditional approaches for identifying and gathering software requirements, including joint application design sessions, requirements management techniques, prototyping, rapid application development, eXtreme Programming, and others. When done properly these techniques typically deliver a form of documented user requirements. After a series of user and peer reviews, these *formal* requirements are typically assumed to represent the complete set of user requirements.

However, the full set of user requirements is generally not complete until the project's end and continues to emerge as it progresses. As a result the project encounters rework, schedule slippage, and budget overruns, the extent of which depends on the degree of originally unknown requirements. With some Department of Defense projects this problem is further compounded due to requirements that are outcome- or performance-based, and functional requirements are developed as part of the design process. When these projects emerge, they challenge traditional requirements approaches. For example, how are software requirements documented when the performance requirement is to launch a projectile from a range of 200 miles with an impact of X? This article is not intended to solve these types of requirement issues.

This article addresses projects where user requirements are articulated (or should be) and outlines how function point analysis (FPA) can be an additional tool to identify missing requirements, gauge requirements completeness, and uncover potential defects. Our experience shows that FPA is often more effective than peer or user walkthroughs in identifying the full set of functional user requirements and uncovering potential defects. In fact, benefits gained by applying FPA to functional user requirements can be more valuable than the mere function point size of the software.

There are two audiences for this article:

1. Development teams that already use or are considering using FPA on their projects. The information provided here is intended to increase the cost-effectiveness of FPA, and leverage its use as a requirements completeness check.
2. Development teams that do not use FPA, but would like additional tools to increase

requirements effectiveness. The concepts outlined in this article can be applied to any project without the need to complete all steps in the method.

Requirements

Why is it that the *right*, i.e. correctly and accurately stated, set of software requirements is so elusive in our industry? Among various reasons, problems involve getting the requirements right, getting the right requirements (the complete set of functional user requirements), and often involve getting more than the specifications of requirements. One of the biggest problems software developers encounter is being able to judge whether requirements are sufficiently complete before beginning formal design and coding.

Before we discuss how to apply FPA to requirements completeness, it is worthwhile to identify the three major types of software requirements. Together, these form the overall project's user requirements. First are functional user requirements, which are the logical business or user functions the software must perform. All software from real-time missile guidance systems to business accounting software has functional requirements that must be performed. These include elementary processes that must be supported to input, process, manipulate, output, and interface data to, from, and within the software. FPA specifically addresses this type of user requirements.

Second are nonfunctional user requirements. These are the technology-independent user/business constraints that the software must meet. Nonfunctional requirements include quality and performance requirements such as portability, usability, security, dependability, reliability, and speed. Part of the FPA technique can assist with these types of requirements.

Lastly, technical requirements are the user requirements for a specific hardware/software configuration or a particular technical configuration that must be delivered. For example, the technical requirements may specify an Oracle database or a multitiered hardware solution. While these software specifications are as important as the other two types, FPA does not address this type of requirements.

The remainder of this article specifically pertains to functional and nonfunctional requirements.

Traditional Completeness Checks

While both the functional and nonfunctional requirements strive to be unambiguous, correct, and complete, it is easy to write down and check business rules for ambiguity and correctness with a user. The problem, however, is to ensure that the full set of functional user requirements has been identified. One or two frames of reference are needed. Such a frame of reference is typically provided using two existing techniques:

1. Theory-Based Model. A theory-based frame of reference may be used as structured analysis, information engineering, or data modeling. The analyst will decompose the problem and look for abstract structures like data flows, processes, and data stores. Requirements will be considered complete when the abstract structures make sense to the analyst, e.g., data stores have both incoming and outgoing data flows.

2. Personal Experience. The analyst may have worked with other business systems similar to the one being analyzed. In that case, he will possess a subjective frame of reference composed of all the business structures and rules he has previously encountered. The analyst will decompose the problem and look for known structures and rules conformant to his own model of reality. Requirements will be considered complete when the identified structures match the analyst's model of completeness, which is subjective, e.g., accounts receivable will have been either received or marked as delinquent.

Ordinarily, the analyst will work with a mixture of the first two frames of reference to increase quality and clarity of the documented set of known software requirements and to increase the relative percentage of the known to total requirements. Throughout the project he will integrate his personal experiences with the theoretical knowledge. Increasingly, however, this is insufficient to gain enough completeness coverage. It is in the analyst's interest to use as many frames of reference as possible.

Ideally, frames of reference should be orthogonal, i.e., they should not overlap. Each frame of reference should provide unique information not available in the other models.

Why Function Point Analysis?

Along with some of the nonfunctional requirements, FPA provides an additional frame of reference for checking the completeness of functional requirements. FPA is different from the first two frames of reference because it provides a unique, user-focused perspective. FPA examines the set of functional user requirements in terms of data and movement/manipulation (transactions) as understood and expressed by users; on this basis, it determines software's functional size. As such, FPA can be used in addition to the theory-based and personal experience models previously mentioned to ensure that functional user requirements are complete.

Function Point Basics

Function points (FPs) measure the size of a software project's logical user functionality as opposed to the physical implementation of those functions as measured by lines of code (LOC). FPA examines the functional user requirements to be supported or delivered by the software. It then assigns a weighted number of FPs to each logical user function as outlined in Function Point Counting Practices Manual [2] and calculates the software's FP size.

In simplest terms, FPs measure what the software must do from an external, user perspective irrespective of how the software is constructed. While analogies from other industries such as building construction and manufacturing attempt to describe how function point analysis works with software, none provides a perfect fit. In basic terms, FPs reflect the functional size of software, independent of the development language and physical implementation.

FPs can be likened to the functional area of a building by summing up its floor plan size. FPs quantify the functional user requirements (the floor plan) by summing up the size of its functional *components*. As with building construction, project management is not possible if only square foot size is known. System development cannot be managed purely on the basis of FP size.[1]

Using FPA to Gauge Completeness

For an introductory article on FPs, see the February 1999 issue of CROSSTALK. When performing a FP count, all the known functional user requirements for the software are analyzed, weighted, and counted using the standard identification method. It is during analysis of functional user requirements that most errors and omissions in the requirements are uncovered, as described below. Following are the steps in the actual FP counting process:

1. Determine the project scope and purpose of the function point count. For example, FPs can be counted to quantify the size of a new development or enhancement/renovation project, or to size an existing base application.

In this step it is useful to document the specific name and date of the source document(s) used as a basis for the count (e.g., system ABC requirements document V1, March 22, 2000). This provides traceability of logical functions included within the functional requirements as a specific point in time, and is useful for gauging scope creep during the project. It can also contribute to the historical base for gauging future projects as outlined below.

By documenting—even in a few lines of text—the project scope and purpose of the FP count, project assumptions are clarified and requirements oversights identified. For example, if the purpose of the FP count is to size the amount of customization required for a commercial off-the-shelf package, the scope will include only the customized functions, not the entire package. This provides a delineation of what is included in the project.

2. Identify the application's logical boundary. This step identifies the functions that the software must perform, together with external users interfaces, departments, and other applications. The application boundary for FP counting is not the same as a physical one. Instead it is the logical boundary that envelops self-contained user functions that must exist to deliver the user requirements. This boundary separates the software from the user domain (users can be people, things, other software applications, departments, and other organizations). Software may span several physical platforms and include batch and on-line processes—all of which are included within the logical application boundary. For example, an accounts payable system would typically be considered one application in FPA, even though it may reside across multiple hardware platforms in its physical installation.

Because each *application* or software system has a separate application boundary (e.g., accounts payable would typically be one application, fixed assets may be another) a project context diagram consisting of several circles denoting various application boundaries is often drawn as a part of the functional sizing process. In cases where an enhancement project renovates an application that has little documentation, this step provides a context diagram that can be used later for communicating with the users about the software system. In a particular client situation, this visual depiction of various application boundaries and interfaced applications opens a discussion of client/server migration of certain applications because our diagrams showed which applications would be affected by the migration of a central application. Because these context diagrams are visual in nature and independent of technology, their review often leads to the discovery of interfaces that were previously discussed, but that are missing from the written requirements.

In addition, this step with subsequent steps, clearly demarcates logical boundaries between user applications. By clarifying which functions lie within which applications, there is less likelihood of a set of requirements being overlooked. For example, if a project team assumes that another application will maintain a set of common data, a review of the context diagram showing the interface to the other application may reveal potential oversights.

3. Count the Data Functions. This step considers internal and external data entities. It consists of:

- Identify, weigh, and count the internal logical files (ILFs). These are the persistent logical entities or data groups to be maintained through a standard function of the software.
- Identify, weigh, and count the external interface files (EIFs) that are persistent, logical entities referenced from other applications but not maintained. Typically these data are used in editing, validation, or reporting types of software processes.

When identifying and classifying the persistent logical entities as internal (maintained) and external (referenced-only), it is helpful to draw circles around the entities and their included subentities on a data model or entity-relationship diagram. If there is no data model or entity-relationship model, one is essentially created in this step by building on the context diagram created in the previous application boundary step.

Note that FPA does not count hard-coded data or any tables/files created only because of the physical or technical implementation. The data step records the number and types of logical data elements if they are known, and if they are not already identified in the requirements. This provides a checklist of data entities to gauge the consistency and completeness of transactional (manipulation of data) functions.

By reviewing the entities, whether on a data model or hand-drawn context diagram, and whether they are inside the application boundary (i.e., to be maintained by the software) or external (i.e., to be referenced only) often clarifies comments. Such comments might include: “Why is that entity external? I thought we needed to be able to update that entity.” These would lead to a discussion that either confirms the original requirements or reveals an inconsistency in understanding and a change in the diagram. When the review is combined with the transactions outlined in the next step, the majority of (potential) requirements mismatches are identified.

4. Count the transactional functions. Use the following:

- External Inputs (EIs) that are the elementary processes whose primary intent is to maintain the data in one or more persistent logical entities or to control the behavior of the system. Note that these EIs are functional unit processes and not physical data flows or data structures.
- External Outputs that are the elementary processes whose primary intent is to deliver data out of the application boundary, and which include at least one of the following: mathematical calculation(s), derive new data elements, update an ILF, or direct the behavior of the system.
- External Queries that are the elementary processes whose primary intent is to deliver data out of the application boundary purely by retrieval from one or more of the ILFs or EIFs.

This step is where the majority of missed, incomplete, or inconsistent requirements are identified. This list provides some examples of the types of discoveries that can be made using FPA:

- If a persistent, logical entity has been identified as an ILF, i.e., maintained through a standard maintenance function of the application, but there are no associated EIs processes, there are one or more mismatched requirements:
  – Either the entity is actually a reference-only entity (in which case it would be an EIF), or
  – There is at least one missing requirement to maintain the entity, such as add entity, change entity, or delete entity.
- If there are data maintenance (or data administration) functions identified for data, but there is no persistent logical entity to house the data (ILF), the data model may be incomplete. This would indicate the need to revisit the data requirements of the application.
- If there is a data update function present for an entity identified as reference only (EIF), this would indicate that the entity is actually an ILF. The data requirements are inconsistent and need to be reviewed.
- If there are data entities that need to be referenced by one or more input, output, or query functions, and there is no such data source identified on the data model/entity-relationship diagram/context diagram, the data requirements are incomplete and need to be revisited.
- If there are output or query functions that specify data fields to be output or displayed that have no data source (i.e., no ILF or EIF), and the data is not hard-coded, there is a mismatch between the data model and the user functions. This indicates a need to revisit the data requirements.

Most maintained entities (ILFs) follow the Add, Update, Delete, Inquiry, Output (AUDIO) convention rule [3]; each persistent logical entity typically has a standard set of functions associated with it. Not all entities will follow this pattern, but AUDIO is a good checklist to use with the ILFs.
5. Evaluate the complexity of nonfunctional user constraints using a value adjustment factor. Through an evaluation of the 14 general systems characteristics (GSCs) of FPA (e.g., the GSCs include performance, end-user efficiency, transaction volumes, and others), a software complexity assessment can be made. The impact of user constraints in these areas is often not enunciated or even addressed until late in the software development life cycle, even though their influence can be major on the overall project.

Examining the user requirements with these nonfunctional, user business constraints in mind can provide the following types of valuable information:

- The nonfunctional requirement due to transaction rate peak loads may necessitate 24-hour, 7-day-a-week availability. This will have a critical impact on the resulting project.
- Special protection against data loss may be of critical importance to the users' business and must be specially designed into the system. This must be identified up front to avoid any unforeseen impact.

FPA provides an objective project size input for use in software estimation equations (together with other factors), or to normalize measurement ratios. The process checks whether the full set of functional user requirements has been identified and can uncover defective and missing requirements. Table 1 summarizes how to use FPA to uncover requirements defects. The far-right column of Table 1 illustrates where and what type of potential requirement problem there might be.

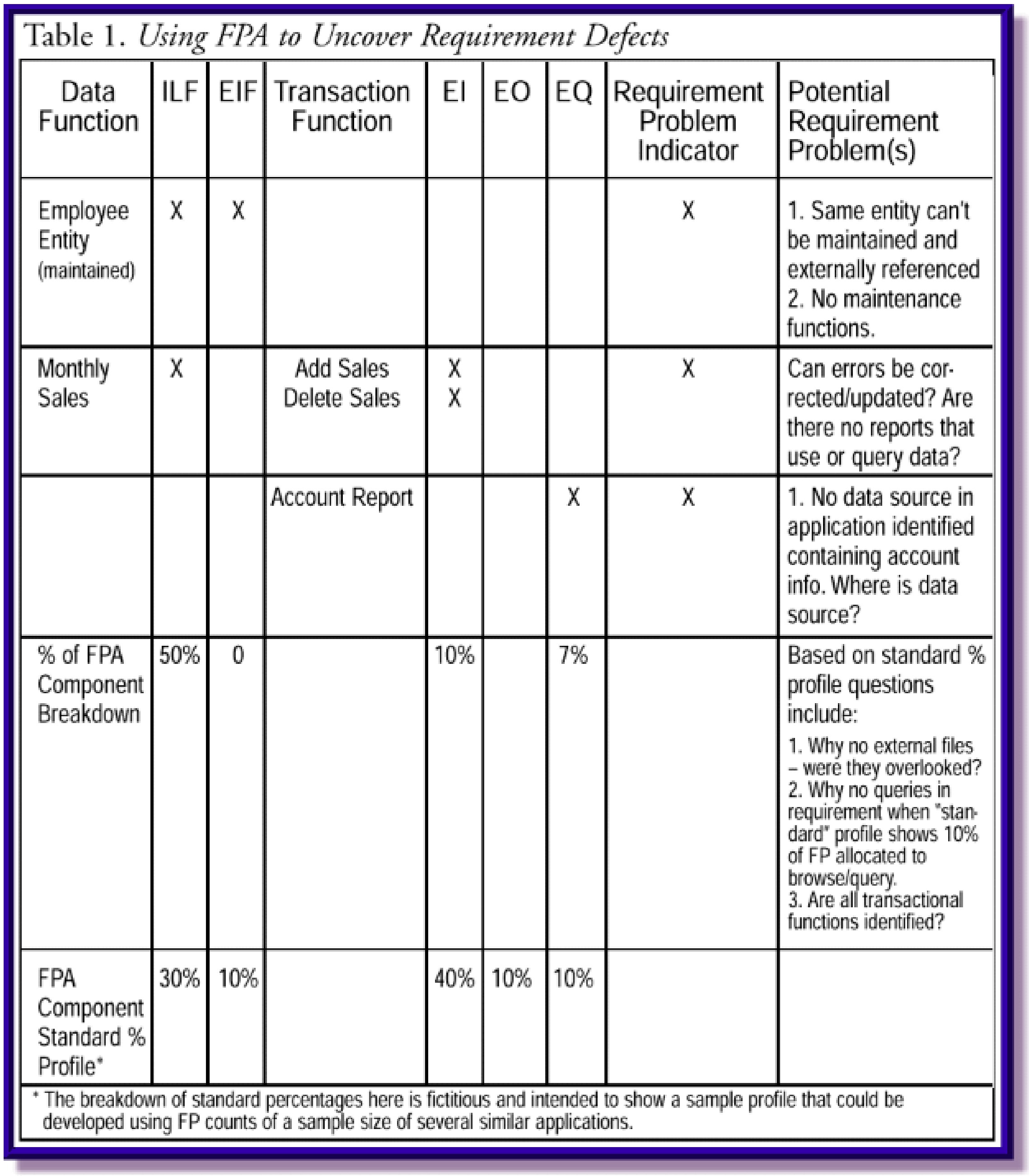


Table 1. *Using FPA to Uncover Requirement Defects*

Data Function	ILF	EIF	Transaction Function	EI	EO	EQ	Requirement Problem Indicator	Potential Requirement Problem(s)
Employee Entity (maintained)	X	X					X	1. Same entity can't be maintained and externally referenced 2. No maintenance functions.
Monthly Sales	X		Add Sales Delete Sales	X X			X	Can errors be corrected/updated? Are there no reports that use or query data?
			Account Report			X	X	1. No data source in application identified containing account info. Where is data source?
% of FPA Component Breakdown	50%	0		10%		7%		Based on standard % profile questions include: 1. Why no external files – were they overlooked? 2. Why no queries in requirement when "standard" profile shows 10% of FP allocated to browse/query. 3. Are all transactional functions identified?
FPA Component Standard % Profile*	30%	10%		40%	10%	10%		

* The breakdown of standard percentages here is fictitious and intended to show a sample profile that could be developed using FP counts of a sample size of several similar applications.

Benefits After the Requirements Phase

Having a documented set of functional user requirements (and the nonfunctional requirements that FPA addresses) such as that provided by the FPA process goes far beyond merely the requirements phase. Hill and Tinker Air Force Bases' Materiel Systems Groups (MSGs) found this to be the case. An example from Hill serves to illustrate this point: The MSG would attach a full listing of functional requirements (using the FPA-documented breakdown of FP components counted) to the software project estimate sent in to headquarters. Later, when questions arose about a particular set of functionality and whether it had been included, the group would refer to the FP listing to see if the particular functionality was listed. If it was not, it was clear that the functionality had not been included. A decision was then made about whether or not to include it and increase the estimate.

This simple set of documented functions minimized the finger pointing and blaming of "who said what and when," and reduced the discussion to whether or not the functions were included in the specifications submitted. Additionally, when scope changes emerged later in the project, as they inevitably do, both groups were in a position to adjust their FPA sizing and quickly assess the impact of scope change on the project.

While other requirements review and tracking techniques can also provide value, FPA is a simple method that delivers both a functional size of the software (useful for estimating) and can assist with the requirements processes.

Summary

Today's software analyst needs all the assistance he or she can find to help in the quest for complete (and known) user requirements. The framework provided by the structure of the FPA technique gives the analyst one extra frame of reference to gauge the completeness of the known user requirements. Requirements defects will still occur no matter how many frames of reference are used, however, the use of FPA to augment the traditional theory-based and personal experience frames of reference will increase the analyst's ability to ensure that software requirements are complete.

Is FPA worthy of your organization's consideration? The answer will vary depending on your organizational structure, goals, and measurement objectives. FPA is one tool that can assist with your requirements processes and also provide a quantitative value to size your software. For those of you who have been using FPA only to arrive at a software size, you can gain valuable benefits by applying FPA as a structured review, especially when your requirements are deemed *complete*.

References

1. Quality requirements can be found in the ISO/IEC 9126:2000 suite of standards that address many of the *ility* constraints such as portability, security, usability, reliability, etc. Contact ISO for further details.
2. The Function Point Counting Practices Manual (CPM) is maintained by the International Function Point Users Group (IFPUG) and is currently in Release 4.1 (1999).
3. Per personal discussions with John VanOrden, certified function point specialist, formerly of Gartner Group and a member of the Quality Plus Technologies Inc. consulting team. VanOrden uses the AUDIO checklist.

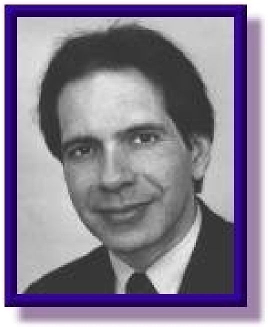
Note

1. When matters of software estimating are discussed, many more factors are involved beyond the functional size of software, including the type of software, technical requirements, number of users, geographic locations, etc.

About the Authors

Carol Dekkers is vice-chair of the Project Management Institute Metrics Special Interest Group. She is president of Quality Plus Technologies Inc., a management consulting firm specializing in helping DoD and private organizations succeed with function points, make wise investments in software measurement, and achieve bottom-line improvements through process improvement. Dekkers is a past president of the International Function Point Users Group and an International Organization for Standardization project editor on the Functional Size Measurement project. She was named one of the 21 New Faces of Quality for the 21st century by the American Society for Quality. She is a professional engineer, certified function point specialist, and a certified management consultant.

E-mail: Dekkers@qualityplustech.com



Mauricio Aguiar is a software manager with Caixa Economica Federal, a leading Brazilian government bank with more than 2,000 branches. He has 25 years' experience in software management, including the application of accelerated learning in information technology. Aguiar is president of the Brazilian Function Point Users Group and serves on the International Function Point Users Group (IFPUG) board of directors. A professional engineer and systems analyst with a master's degree in neuro-linguistic programming, he is a member of Project Management Institute, American Society for Quality, and the IFPUG.

E-mail: mauricioaguiar@yahoo.com