12 The IFPUG Function Point Counting Method

The IFPUG (International Function Point Users Group) Function Point Method (FPM) is a method to measure the (functional) size of software from the user perspective (depicted in Fig. 12.1).

Functional size is defined (according to ISO/IEC 14143-1:2007 Software and Systems Engineering – Software measurement – Functional size measurement – Definitions of concepts) as: "a size of the software derived by quantifying the Functional User Requirements," where the *Functional user requirements* (FUR) are in turn defined as a subset of the User Requirements. Requirements that describe what the software shall do in terms of tasks and services.

As an ISO/IEC conformant Functional Size Measurement (FSM) method, the IFPUG FPM measures the functionality in software delivered to the user as required by the user, and quantified by following the IFPUG Counting Practices Manual (CPM) set of counting rules. Note that functional size is purely the unadjusted function point size as outlined in the following paragraphs.

The term *user* is not defined strictly as a person or end-user, but rather as any person, thing, other application, hardware, or software that needs to interact (send to or receive data from) with a piece of software. This is consistent with the term *actor* in object-oriented or use case technology.

The functional size measure is independent of the nonfunctional requirements, including the technology used for implementation, since the technological aspects of the software development are not part of the functional size.

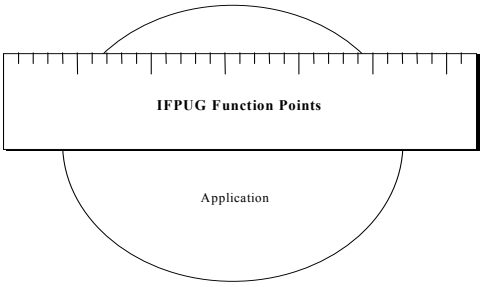

Fig. 12.1. Counting a software application

Function Points are derived from the logical (or functional) user requirements concept, and the person counting the Function Points will learn a lot about the functional requirements of the software during the process of evaluating the functional size.

The following goals are often cited for using the IFPUG FPM:

- Standardized and integrated software measurement
- Improvement of estimation accuracy and project management
- Improvement of quality of the development process
- Knowledge transfer of estimation experiences and lessons learned
- Reduction of complexity and uncertainty in the estimation process (because the object of estimation has been quantified as part of the sizing process)
- Basis for indicators and metrics.

An extract of the exact rules for counting according to IFPUG are provided in a further chapter of this book. There exists a wide variety of information sources about the IFPUG FPM on the Internet; however, the actual IFPUG website is http://www.ifpug.org. Note that IFPUG is a not-for-profit users group headquartered in Princeton, NJ, and it is owned and operated by and for the members. IFPUG is not associated or managed by any vendor or consulting organization.

Additional sources of IFPUG methodology information (not all of it in accordance with the official IFPUG function point counting practices) include the following:

- Carol Dekkers, Quality Plus Technologies, Inc.: http://www.qualityplustech.com,
- David Garmus: http://www.davidconsultinggroup.com,
- The IT Metrics and Productivity Institute: http://www.itmpi.org/
- The University of Quebec at Montreal, Canada: http://www.lrgl.uqam.ca,
- Capers Jones and Software Productivity Research: http://www.spr.com.

During the last decade, many advances have been made worldwide to popularize and advance the use of FSM of software. In particular, five of the frequently used ways of sizing software are conformant with the ISO/IEC definitions and themselves have become ISO/IEC standards. All FSM methods evaluate software based on its functional user requirements. This means that the functional size is independent of the development environment and user demands for quality; in other words, the functional size does not change with changes in development technology, programming language, skills, experiences, or performance of the developers. They also agree that functional user requirements can be defined in a catalogue of logical transactions that will be performed by the software and countable in functional size measurement units.

The current ISO/IEC conformant FSM methods are:

- ISO/IEC 19761:2002 Information technology, Software and systems engineering – COSMIC-FFP: A functional size measurement method
- ISO/IEC 20926:2002 Information technology, Software and systems engineering – IFPUG 4.1 Unadjusted functional size measurement method: CPM
- ISO/IEC 20968:2002 Information technology, Software and systems engineering – Mark II Function Point Analysis: CPM
- ISO/IEC 24570:2004 Information technology, Software and systems engineering – NESMA functional size measurement method version 2.1: Definitions and counting guidelines for the application of Function Point Analysis
- ISO/IEC 29881:2008 Information technology, Software and systems engineering – FISMA 1.1 functional size measurement method.

12.1 Functional Size Measurement Methods History

The first method that described function point analysis was originally developed in 1979 by A. J. Albrecht from IBM, and first presented publicly at a GUIDE/Share conference. Interest in the method and its application as a basis for objective estimating grew quickly around the world, and by 1986, the IFPUG was formed in Toronto, Canada. The first IFPUG CPM version 1.0 was released in 1988 by the IFPUG. IFPUG 1.0 (as it is abbreviated in general usage) formalized Albrecht's 1984 standard set of function point rules. Since its inception, IFPUG has remained a volunteer, not-for-profit membership organization based in the United States, with members residing in many of the countries where software process improvement and measurement are important. Membership benefits of the IFPUG include the CPM in its current release as well as reduced conference attendance and discounts on publications (see http://www.ifpug.org). The CPM describes the counting rules in a standardized form. Since the first CPM appeared nearly 20 years ago, the IFPUG FPM has been translated into German (IFPUG 4.0), French, Italian, Japanese, Korean, Portuguese, and Spanish. The IFPUG publishes biannually the M*etric Views* – not to be mistaken for the biannual Germany journal: *Metrics News* (also in English, older editions downloadable free of charge) of the German GI Metrics group (http://ivs.cs.uni-magdeburg.de/sw-eng/us/ - the German Metrics News actually changed its name in 2008 to Software Measurement News).

The IFPUG Standard 4.1 is acknowledged as an ISO/IEC Standard 20926, and to conform to ISO/IEC definitions, it had to be published with the 14 GSCs (General System Characteristics for calculation of adjusted Function Points) being *OPTIONAL only*. (ISO/IEC defines "functional size" as describing only the tasks and business processes supported by the software, and the 14GSCs and resultant VAF go beyond mere functional size). The formal release of IFPUG CPM Release 4.2 (2004) included the 14 GSCs as a mandatory step in

IFPUG function point counting; however, the next version of the IFPUG standard when it is submitted to ISO/IEC to replace the ISO/IEC 20926:2002 standard will again reference the Value Adjustment Factor (VAF) as an optional step to conform with the ISO/IEC definitions.

Figure 12.2 shows the evolution of the functional size measurement Methods from IFPUG, COSMIC-FFP, NESMA, FiSMA and Mark II during the last 30 years.

The software functionality measured by the IFPUG counting practices manual rules is clearly based on an elementary process-oriented, stimulus-response-model. This implies that the composite counting items inputs, outputs, and inquiries are each transactional types that interact (receive or send data) in relation to a "user" (as previously defined). This was not so clearly stated in earlier IFPUG CPM releases, and the lack of clarity hindered the usability (and potential applicability) in the past.

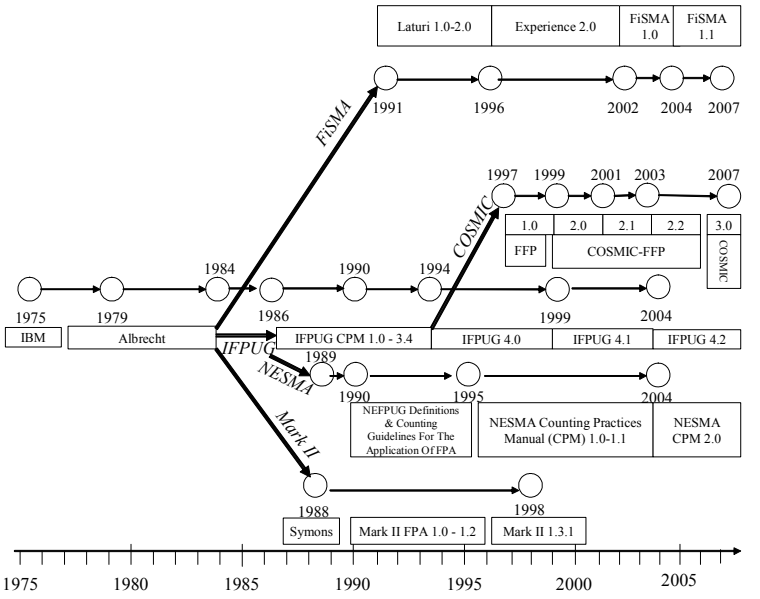

Fig. 12.2. History of Functional Size Measurement (FSM) methods

12.2 The Benefits of the IFPUG FPM

The IFPUG FPM can easily be learned and understood and applied to a variety of software. This becomes important particularly when counts may have to be audited or formally released as part of the quality assurance process. Function Points define objectively the functional size of software applications from the

user view, and are typically expressed in the user's language. This is consistent with the understanding about the functionality of their application. The fact that a function point count must be done based on the functional user requirements has the added bonus that it forces the project team to see the software from the perspective of the user and to respond accordingly. The functional elementary processes should identically match the specified functional user requirements. A by-product of function point counting is a better design and improved control during the project.

Authors' note: It has been observed through first hand experience that when the Function Point Analysis is done with the involvement of the end users, they are motivated to better teamwork and more committed engagement. In addition, we have observed that the overall user satisfaction with the project increases. This is a key project success factor according to the Standish Group's annual CHAOS Report.

12.2.1 Leveraging the Use of Function Points (and the Analytical Approach) in Software Development

Figure 12.3 shows how measured Function Points can profitably be leveraged for the software engineering process and project management groups.

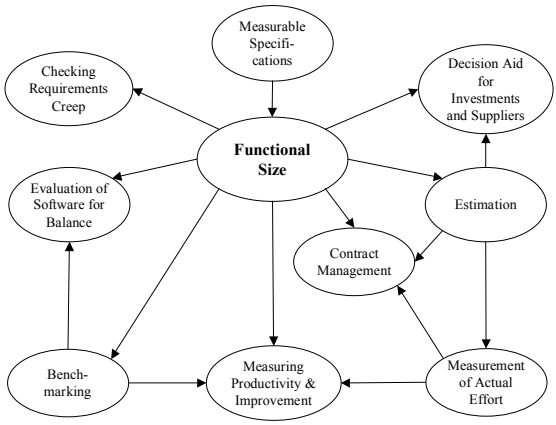

Fig. 12.3. Leveraging Function Points to benefit software development

The following list includes ways that Function Points (FP) can be used in the short term.

- FP size as input for estimation and project management.
- FP list of included functions as the basis for project planning and architectural design.

- FPs allocated to parts of the software as the basis for structuring projects and planning of releases.
- FPs size as a basic metric for quality-planning and management-reviews (common denominator for defect density).
- FP methodology as implicit inspection of requirements for completeness and misunderstandings, and quality improvement of user-specified requirements.
- FP methodology for design of test cases and for estimation of test effort.
- FP size contributes to metrics for stability and reliability (Mean time to failure as a function of size).
- FP size as the basis for software benchmarking and risk analysis.
- List of functional user requirements (on which the count is based) delivers user-oriented documentation of the application.
- FP size at various points in the development life cycle is used for measurement of requirements creep (scope management).
- FP size as one of the input variables for calculation of various productivity and quality metrics.
- FP methodology supports reuse in IT development by early and standardized quantification of business cases in the requirements definition phase, for contracting, for project-estimation, for test case identification, for enhancements, and for documentation.

Figure 12.4 shows the benefits of the FPM at a glance.

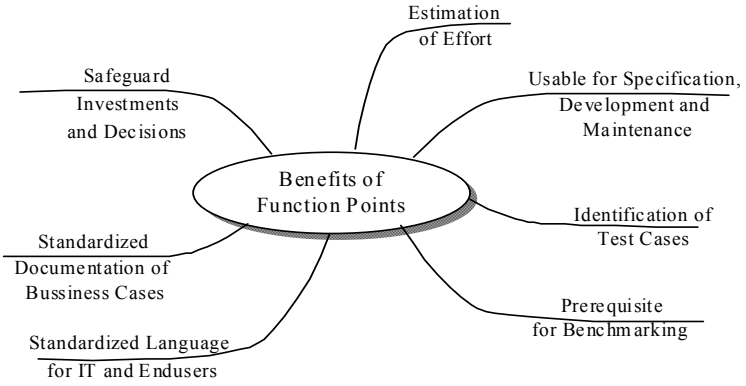

Fig. 12.4. Mind map of benefits of the Function Point Method (FPM)

The principal benefits of the *FPM* include the following:

- The methodology is independent of the development environment as well as the skills or attributes of developers.
- By consequent use of Function Points according to the IFPUG standard and careful documentation of the counting results, the organization gets valuable

interproject consistent data about the size of elaborated and to be developed applications. This information is *the basis for solid effort estimates* for software development.

- Through internationally agreed standardizations based on functional size, *interorganizational benchmarks can be enabled*.

The FPM facilitates estimation in an manner easier and more precise than other assessments of size (such as the use of unqualified judgements of small, medium, or large software size). Function Points can be used during specification, development, enhancement, and maintenance of software, as well as for safeguarding investment decisions. Beneficial side effects are quantified quality (when FP are correlated with defects), risk awareness, easy to be derived test cases, measurable productivity (when correlated to effort hours), and standardized business requirements (for users as well as for developers).

Figure 12.5 shows the areas for application of the FPM.

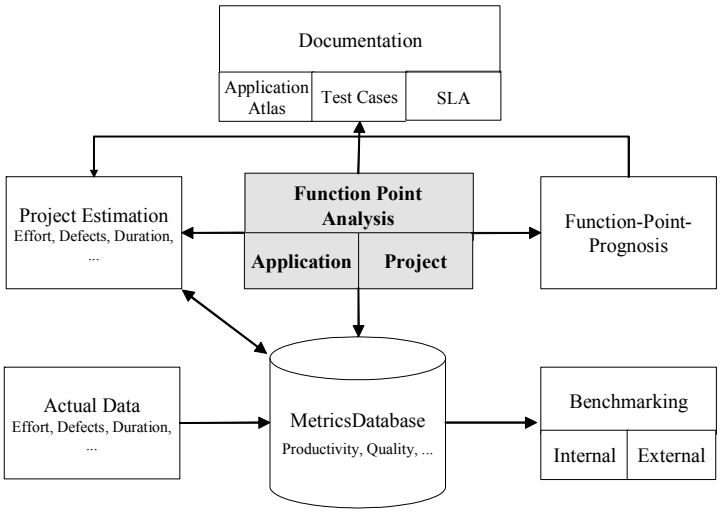

Fig. 12.5. Areas of application of the FPM and/or functional size

12.2.2 Function Points as Part of a Pricing Evaluation Process

Function Points can facilitate comparison of prices from suppliers and to evaluate cost ratios for software under contract (e.g., price per FP or FPs per US-$ or per Euro). This metric (cost per FP) can also be discussed with the software suppliers. Capers Jones reports in *IT Measurement – Practical Advice from the Experts* (IFPUG 2002) that "standard" software such as spreadsheets

can be bought for about 0.25 US-$ per FP. Specialized niche products may then cost about 10 to more than 300 US-$ per FP. Development costs of applications may vary widely from a low of 200 US-$ per FP for small systems to more than 5,000 US-$ per FP for large military or defense systems. These prices can be compared with development costs of about 1,500 US-$ per FP in Western Europe and about 350 US-$ in Eastern Europe.

Howard Rubin also contributed a chapter about pricing comparisons in the above mentioned book.

ISBSG publications also discuss how function points can be used as part of price comparisons (see http://www.isbsg.org). The metric "price per FP" can contribute to decisions about whether to "build-or-buy." Note: Build-or-buy is an English expression meaning a decision about "Building" customized software, typically under contract with a supplier; or "Buying" standard packaged software. Figure 12.6 shows how Function Points fits into the decision making process associated with Build-or-Buy decisions.

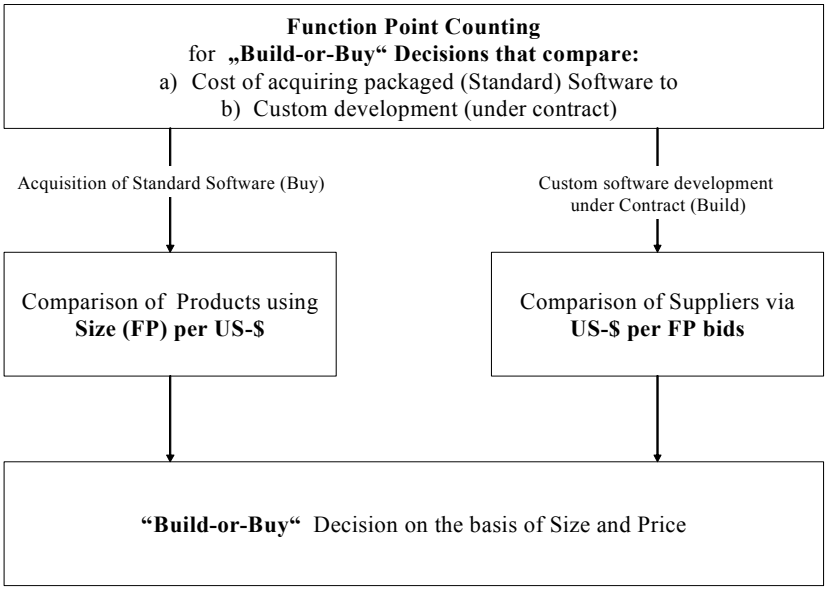

Fig. 12.6. "Make-or-Buy" decision based on using FP size as part of the pricing equation

12.2.3 Function Points as the Basis for Contract Metrics

Another interesting use of Function Points is to directly measure and manage software development performed under contract. Because FP are independent of the tools, techniques, people, and the technical implementation of

software, different perspectives between a purchaser and supplier can be discussed objectively on the basis of Function Points. Capers Jones reports in *IT Measurement –Practical Advice from the Experts* that between 1995 and 2001 FPs and LOC metrics were in direct conflict in at least a dozen U.S. Internal Revenue Service (IRS) cases, with the LOC metric being on the losing side of the judicial decision in virtually every case. In one tax case in 1996, both IRS and the defendant used FPs to prove their case. High profile precedents like this helps to resolve any anxiety practitioners may have about the unreliability of software measurement and estimation.

A few examples of contract metrics include the following:

- Contracted price per delivered FP for new development or enhancement.
- Fixed price for a certain number of FPs (new development or enhancement), which can then be normalized to a cost per FP.
- Fixed or FP based pricing for the maintenance of a portfolio with a certain size as measured in FPs (typically done on the basis of cost per 1,000 FP).
- Variation from fixed price (at preagreed cost per FP) if the software size is larger or smaller than negotiated.
- For improvement of team or departmental performance (setting goals, bonus systems) the evaluation may be based on the following:
    - Delivery rate expressed as number of FPs per hour for new development or enhancement
    - Cost per FP for new development or enhancement
    - Maintenance load expressed as number of FPs maintained per person in 1 year (person is often referred to as "Full-Time-Equivalent" or FTE in North America)
    - Defect density expressed as a number of defects per FP
    - % improvement in delivery rate based on comparison between a current and previous delivery rate.

Using function points as part of performance measurement in contractual arrangements, the following measures are generally collected:

- Number of FPs for each project or application (depending on what performance metric is desired: project FP are needed for productivity metrics; application FP are needed to determine support ratios)
- Price points for delivery of different types of development or for different levels of FPs (price per FP)
- Estimated costs, effort hours, duration, anticipate team size (by job role and availability)
- Actual costs, effort hours, duration, actual team size (all at project postmortem)
- Tracking of project progress and measures in the case of delay (together with mitigating factors for delay)
- Approved changes (sized in FP).

In summary:

The effort for planning, performing, and documentation of Function Point counts can be justified as long as FP are used appropriately together with other measures. Function points in and of themselves tell only the functional size of software in a manner similar to the square foot (or square meter) size of a building tells only the area of a floor plan. When used appropriately in performance measurement, function points provide an objective denominator (as in a "per square foot") that normalizes metrics for comparison across software projects. The benefits in such cases far outweigh the costs of the learning curve and organizational resistance, and additionally the structured analytical approach to counting FP provides intangible gains to the requirements process. Capers Jones estimates the effort to implement a fully-functioning measurement and analysis program to be maximally 3% of the cost of a project – not much when you consider the savings that better requirements and accurate estimates can provide.

12.3 Application Areas for Function Points

The primary application areas of the IFPUG FPM are in estimation of new software development and enhancement. The following is a partial list of the most common application areas for FP-based metrics:

1. Estimation of software development costs and/or effort (based on FP and other project attributes)
2. Estimation of maintenance costs and/or effort of implemented systems (based on FP supported per person figures)
3. The Earned-Value method has been applied to some projects based on Function Points for evaluation and delivery. More research is needed in this area. Capers Jones addresses this topic in *IT Measurement – Practical Advice from the Experts*
4. Comparison of functionality of an old system vs. its replacement during reengineering (rebuild)
5. Projection of productivity trends in software development based on historical rates (FP per hour for particular types of development)
6. Cost estimation based on cost per FP or FP per hour (speed of delivery) as a basis for planning of resources and milestones
7. When parts of a project have to be delivered in releases, the functionality can be allocated to and accounted for using FPs
8. Defect density metrics (defects per FP) can be used for better planning of the test phase in the project

9. *FPs can be used for risk assessment.* Capers Jones published in Assessment and Control of Software Risks that projects with less than 500 FPs fail only in 20% of all cases, whereas the failure rate of projects with more than 5,000 FPs is about 40%
10. Pam Morris (in *IT Measurement – Practical Advice from the Experts*) found with regression analyses a correlation ($R^2 = 0.8638$) between the size of an application measured in FPs and the number of persons (P) necessary for maintenance:

$$P = 0.0012 \times \text{FP}.$$

Thus, for the maintenance of an application with a size of 1,000 FPs there are 1.2 persons necessary or 1 person per 833 FPs.

It has been suggested that the applicability of the IFPUG FPM is restricted to *commercial applications* (Management Information Systems, MIS) for the reason that the development costs for engineering or other types of more complex applications depends from other factors. (Commercial applications mainly manipulate large data volumes and use many inputs and outputs.)

Technical or scientific applications (e.g., in R&D or production) focus mainly in processing of data, and often involve complex calculations and combinatory problems to be solved. While the IFPUG methodology does not regard these aspects explicitly, it should be noted the factors influencing work effort and cost are explicitly external to any functional size measurement method as defined in ISO/IEC 14143-1:2007 Functional Size Measurement – Definition of Concepts.

For more technical IT projects there is a stronger orientation on processing criteria. In this area input and output functions are often trivial, whereas processing features have an important role. The *COSMIC Method*, presented in the chapter about *variants of the FPM,* claims to address internal processing more concretely, and the reader is directed to select the most appropriate functional size measurement method (amongst the five ISO/IEC conformant methods) to meet their specific needs. The one caveat is that it is usually best to select one method for all of your functional sizing needs so that the functional size of various projects can be effectively and easily compared.

Chris Kemerer from the Massachusetts Institute of Technology (MIT) showed in a research study comparing 15 software projects that the FPM could be used for various types of software beyond management information systems or commercial applications. Furthermore, Kemerer found that estimation based on functional size measurement produced the most consistent and accurate results compared to source-lines-of-code (SLOC) based estimating methods (SLIM, COCOMO, and Estimacs).

12.4 The Evaluation of Function Point-Based Estimation Methods

Noth and Kretzschmar (in their book in 1984) tested 20 different methods of estimating software development effort, and found that those based on sizing with Function Points belong to the few methods that they could recommend for use. This can be seen from their test protocol of the FPM shown in Table 12.1.

According to Noth and Kretzschmar, using function points (functional size measurement) as the input variable for size in effort estimating has the following advantages:

- The size measurement focuses on the functional size, which was regarded as the best option.
- By using a specific organizational experience curve, many different influences can be combined in a single formula and then the particular individual influences do not have to be separately examined in detail.
- The estimating method can easily be adapted according to organizational requirements.

Table 12.1. Test protocol from Noth and Kretzschmar (1984) of estimating models based on size measurement using Function points

Test criteria	Evaluation	1	2	3	4	5
Ease of use	Usability			x		
	Ease of learning				x	
	Effort to develop estimates from the measure				x	
	Tool support		x			
	Transparency	x				
Contribution to project control	Applicable early in the development life cycle	x				
	Structuredness				x	
	Ease to apply iteratively	x				
	Sensitivity analysis				x	
Quality of results	Precision	x				
	Understandability	x				
	Ease of evaluation		x			
	Degree of influence	x				
	Number of parameters			x		
	Objectivity				x	
	Stability		x			
	Defect localization		x			
	Adaptability	x				
	Adaptivity	x				

1, excellent; 5, poor

Noth and Kretzschmar say that the biggest disadvantage of function-point-based estimating methods is that a detailed estimation on an individual module-by-module basis is not possible. *As such, the method is only applicable for gross planning.*

Another evaluation of function point-based estimating was published by Ruede who used the catalogue of criteria from Herrmann:

- *Precision:*
  A high degree of precision can be achieved by the transfer of experiences between the project postmortems and subsequent new development projects. Functional size measurement-based estimating delivers more precise results over the course of usage.
- *Standardization:*
  Functional size (also known as a function point count) can be easily understood by an end user with respect to its content and basic calculations.
- *Early Applicability:*
  Functional size measurement can be used very early since the requirements are the basis for the counts. See also the chapter about Function Point Prognosis in this book.
- *Data Collection:*
  The necessary information for estimation and functional sizing can easily be gathered.
- *Objectivity:*
  Functional size is not influenced by demands from management or individuals.
- *Transparency:*
  Functional size measurement can be done together with the end user. The resultant estimates based on this size can be explained and controlled easily.
- *Degree of Details:*
  The effort for single activities or tasks and to develop specific programs and modules cannot be evaluated using FP-based estimating methods, rather the effort for the lifecycle development of software applications and projects. A detailed view is possible from the user side.
- *Stability:*
  Functional size results are stable even when development techniques or methods change.
- *Flexibility:*
  Functional size deviations can easily be seen during the iterative process and comparisons made between the planned functionality vs. what was actually delivered. Evaluations can be corrected.
- *Ease of Use:*
  Functional size measurement can easily be learned, the number of parameters on which it is based is acceptable, and the sizing process is not time-consuming.

Hence, estimating methods based on the FPM for sizing software fulfill the major prerequisite requirements of a method for estimation of effort.

Besides the test criteria, Ruede published two other essential advantages of using functional size-based estimation:

1. Software development evolves in the direction of IT organizations and neglects the areas of programming replaced by tools. FPMs are the correct way to assess software functional size since the user requirements are the basis for calculations.
2. The productivity of application development can be demonstrated and improvements can be planned based on per FP calculations where the common denominator is functional size.

The above-mentioned study by Kemerer shows also that Function Point counts performed using the same functional size measurement method and release (e.g., IFPUG release 4.2) can be compared between different organizations (benchmarking).

Many users choose to employ size measurement using IFPUG or other Functional Size Measurement Method because of its early applicability in the software life cycle, and also because it delivers objective and consistent estimates of functional size even when requirements are not concrete. Through using FP as the common denominator (similar to using per square foot or per square meter ratios in building construction), function point based estimating also delivers the chance to gain experiences and rules of thumb.

The goals of the IFPUG function point analysis method are to measure small units in order to support flexible comparisons and early deviations from plan. A basis for planning can be elaborated and the controlling of IT projects can be improved. More precise estimates of size (and effort using a FP-based estimating model) for follow-up projects are possible, and effects of changes in the development environment become transparent.

It is important to note that there are obstacles to the universal application of FP-based estimation and functional size measurement, including wide spread prejudices (and ignorance due to misunderstanding or lack of "informed" opinions), leading to the conclusion in some circles that they are not feasible or even that they should be avoided. Figure 12.7 provides some counter-arguments for the types of statements that are often levied against functional size measurement.

12.5 The Optimum Time to Count FPs

The optimum time for a first Function Point count is the end of the requirements analysis. This phase delivers the following:

- Description of the user requirements
- Description of the data structures.

This contains all necessary information for a Function Point count.

The chapter *Estimation Fundamentals: The Right Time for Estimation* at the beginning of this book is also valid for Function Point counting. Note that before this point in the development life cycle, function points can only be *estimated* but not counted.

It does not make sense that a Function Point count is only performed once during project progress, because throughout the project there evolves new information such as scope changes, clarifications to requirements, (as well as requirements creep).

We recommend (consistent with the practices at IBM) to revisit the original Function Point count for any updated information (and changes) at the end of each phase of the software project. This practice also supports the tracking of requirements scope creep and scope management principles.

The counting of Function Points is ideally considered to be a part of the project documentation, reviews, project controlling, and releases at the end of each project phase.

Revisiting the Function Point count at different times as the project progresses enables early adjustments (and corrections) to the resultant effort estimates, and thus it increases the precision and approximation of the actual effort.

Points and counter-points about function points...

PREJUDICE (POINT)	COUNTER-POINT
... they are developed by theoreticians or academicians and they are not practical for use.	☺ Originally developed by A. Albrecht as a in-practice project for the development of system software at IBM.
... They produce administrative overhead.	☺ The effort to perform FP counts compared to their benefit, and the overall project effort is negligibly low (less than 3%).
... They are not usable for object-oriented or other types of application development.	☺ FP's are a Meta-Model that allows a mapping of the functional requirements, no matter in which description or technical implementation is used.

Fig. 12.7. Counter points to prejudices against Function Points

When an effort estimate is required at an earlier stage than at the end of the requirements phase, we recommend the development of one or more Function Point prognosis methods. See the chapter titled *Application of the FPM: Function Point Prognosis* in this book. This requires counting of historical, completed software projects and requires one to perform regression analyses. Experiences from a sample size of 16–20 completed projects can form a reliable basis for such methods. Another approach is to use the SPR-Function Points (see chapter *Variants of the FPM: SPR Function Points*) when you need a FP estimate before completing the requirements phase.

12.6 The Process of Function Point Counting

Before starting a Function Point count using the IFPUG method, the following information must be available to the counter:

- The outputs produced by the *application*
- The inputs entering the *application* across its boundary
- The internal logical files that are maintained by the *application*
- Entities and Relationships between internal logical data
- Inquiries for data retrieval that can be asked of the *application*
- Interfaces between the *application* and other *application*s
- Interfaces between the *application* and its *users*
- Key *logical* processes of the *application.*

Note again that the word *user* in function point terminology means *anything (i.e., human users, other applications, hardware, software, etc.) that interacts with the software.* This is similar to the word "actor" in use case terminology.

The process of Function Point counting is described by IFPUG as follows (see Fig. 12.8):

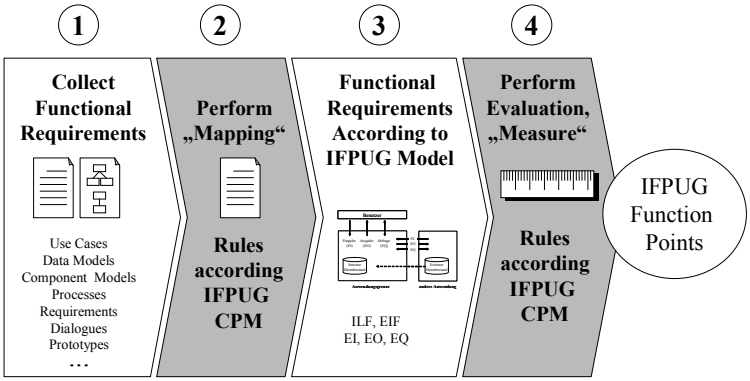

Fig. 12.8. The process of IFPUG Function Point Counting

- Define the type of count
- Define the scope of the count and the system boundary
- Count the unadjusted FPs.

Note that this is now the functional size of the software according to ISO/IEC where functional size is defined as the size of the functional user requirements. Therefore, the functional size of a piece of software equals the UNADJUSTED Function Point count. However, the next two steps, which have been part of the IFPUG method since the beginning, adjust the functional size by considering the effects of some nonfunctional requirements.

It is anticipated that all future releases of the counting practices manual (IFPUG CPM) will include the VAF (steps 5 and 6) as optional to be consistent with the ISO/IEC version of the IFPUG standard.

Optional steps (in the ISO/IEC version of the IFPUG standard, currently still part of the IFPUG CPM 4.2):

- Calculate the VAF after determining the 14 GSCs
- Calculate the adjusted FPs.

We recommend two further steps (regardless of whether steps 4 and 5 are done) that go beyond the IFPUG rules:

6. Document the details of the count.

Function Point counts as well as FP estimates should be performed by project leaders or project team members knowledgeable about the functionality together with support of the competence center. The release of counts and estimates will be more consistent when there is a final quality check done by the competence center. Thus, our recommended final step is:

7. Quality assurance of the FP count by the competence center.

12.6.1 Step 1: Define the Type of Count

There are three types of Function Point counts, the first two specific to IT projects:

1. New development
2. Enhancement
3. Application.

The relationships between these count types are shown in Fig. 12.9.

A *new development* project is the first build of an application. Thus, all delivered functionality is considered to be added. Thus, Function Points counted are the added (= delivered) plus any FP for user required "conversion" functions.

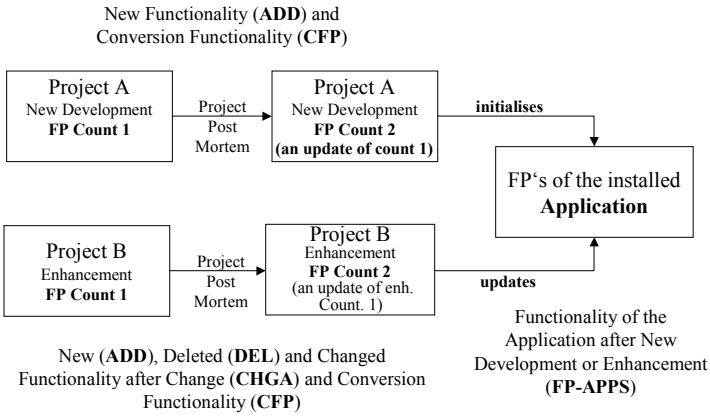

Fig. 12.9. Types of Function Point counts according to IFPUG

An *enhancement* project can add functionality, as well as change or delete functionality. Accordingly, the FP is the summation of the added, changed, and deleted FP, plus any FP for the user required "conversion" functions.

Figure 12.9 demonstrates that the IFPUG methodology regards requirements scope creep based on at least two Function Point counts. The first count will be at project start to measure the planned functionality, while the second will be at project postmortem to measure the actual delivered functionality.

At project postmortem of an enhancement project, the Function Points of the enhanced *application* must also be updated based on what has been added, changed, and deleted.

In the IT department of an international insurance company in Germany, the following standards (see Fig. 12.10) were introduced for definition of an application software (AS):

- Has at least 1 user
- Has at least 1 EI and 1 EO/EQ
- Has a sovereign data-storage, -administration, and -derivation, that is, it has at least 1 ILF (usually, although not necessarily)
- Interfaces must exist to satisfy the logical (functional) user requirements
- Processes business cases completely. Note: the exception is if there are central interfaces or comparable follow up processes involved in the final handling of a process
- Is maintained and administered by one organizational unit (this was a specific internal company standard)
- Different products do not necessarily lead to separate boundaries between ASs
- Inventory or insurance administration will normally be considered as different ASs (specific internal company standard)

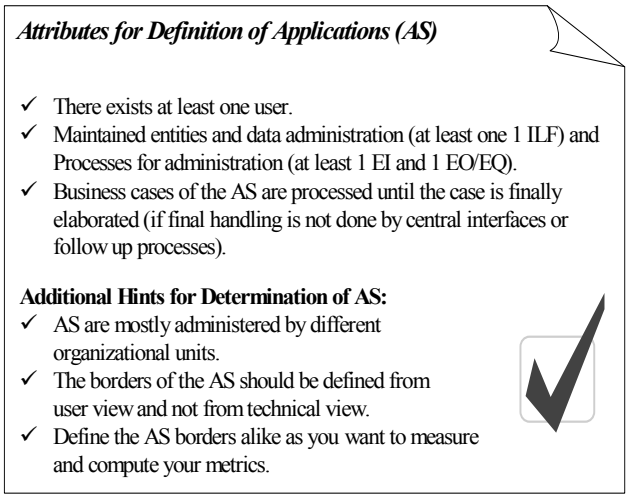

Fig. 12.10. Example of an internal corporate standard for the definition of application software

- Statistical reporting of an administered internal file alone is no reason for definition of an AS
- Batch- or interface-processing vs. online processing should not determine the placement of the application boundary
- Different ASs have unique and different functionalities
- Different users may provide a hint of different ASs, except in the case of interface systems.

Business processes of different mandatory authorities are typically administered in different databases. Exception are if there exist also common processes besides the separate processes in the same database. In these cases they are not defined as separate ASs (this is an internal corporate standard as an example for the readers).

12.6.2 Step 2: Define the Scope of the Count and the Application Boundary

*The IFPUG FPM distinguishes between the size of a software project (Counting scope) and an application.*The size of a project can include several applications each having different functionality from user view (not from technical view) and, thus having different application boundaries. As such, there may be several Function Point counts within a single "business" project.

The definition of the application boundary determines which functionality is counted for the project and which functionality would be counted for external applications.

Some estimators guess that the Function Points have been counted worldwide for about 30,000 applications, but the actual number of discrete software applications is not known other than it is a minority of the actual number of total software applications in existence. As functional size measurement increases in usage, hopefully more than 1% (according to statements by Prof. Alain Abran at the Software Measurement European Forum in 2005) of software organizations will be involved in software measurement.

Principally, the application boundary must be defined from the user view. As depicted in Fig. 12.11, the user is outside the system. After determining the boundary, data files maintained within the application and the associated maintenance functions (create, add change, delete) are counted as internal logical files, with external data files counted for those entities administered and maintained outside the application boundary. In enhancement projects, it has to be regarded that the new application boundary is consistent with the boundary of the base system.

Since the application boundary is critical to the determination of the application functionality, it is important for it to be documented clearly. This includes the description of assumptions used to locate the boundary.

Practically, this documentation (typically including system diagrams) can easily be reused in (or as) architecture diagrams in the application atlas of the organization.

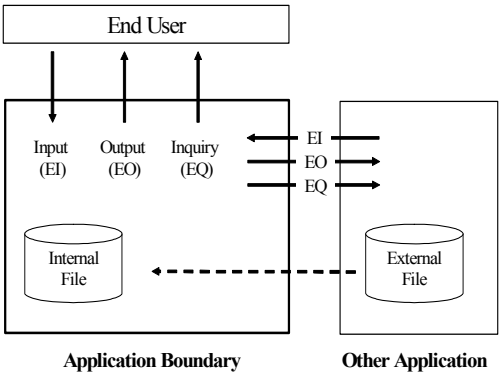

Fig. 12.11. Defining the application boundary

12.6.3 Step 3: Count Unadjusted FPs

The IFPUG Function Point Methodology distinguishes five function types as shown in Table 12.2.

Table 12.2. IFPUG function types

Data function types	*ILF (Internal Logical Files):* Internal logical files with their records and data elements; data that are maintained within the system boundary by the software under consideration. Persistent logical entities *EIF (External Logical Files):* External interface files with their records and data elements; data that are maintained outside the system boundary (by other applications). Persistent logical entities maintained by another application, but referenced by this one
Transaction function types	*EI (External Inputs):* External input functions with their logical data groups and data elements. External inputs are elementary processes *EO (External Outputs):* External output functions data with their logical data groups and data elements. External outputs are elementary processes *EQ (External Inquiries):* External inquiry functions with their logical data groups and data elements. External inquiries are elementary processes

Function Points are counted according to specific IFPUG formulae according to the type of count (see step 5):

- *For enhancement projects:* added plus deleted plus changed plus user required conversion functionality must be counted.
- *Added functionality* enlarges the functional size of the project and the functional size of the base application.
- *Deleted functionality* enlarges the functional size of the project (since it is worked on it), but reduces the functional size of the base application.
- *Changed functionality* enlarges the functional size of the project and can enlarge, reduce, or leave unchanged the functional size of the base application.
- *User required conversion functionality* is counted as part of the functional size of the project, but does not affect the functional size of the base application.

The Function Points are then classified according to a *complexity matrix* into low, average, or high. The result is documented in a Table 12.3.

The sum of the Function Points are called the *unadjusted Function Point count.* This is the functional size according to ISO/IEC. The steps 4 and 5 of the IFPUG method as defined earlier modify (adjust) this unadjusted Function Point count based on the influence of fourteen nonfunctional user requirements.

Note that in the future IFPUG releases (after IFPUG 4.2) it is anticipated that these steps will be deemed to be "optional" steps for consistency with the ISO/IEC version of the IFPUG method.

Table 12.3. Summary of a Function Point Count

Functional type	Complexity	FPs	Number of unique functions	Sum FPs
ILF	Low	7		
	Average	10		
	High	15		
EIF	Low	5		
	Average	7		
	High	10		
EI	Low	3		
	Average	4		
	High	6		
EO	Low	4		
	Average	5		
	High	7		
EQ	Low	3		
	Average	4		
	High	6		
	Sum of unadjusted FPs			

The next step involves determining the influence of the 14 GSCs. The sum of the values for the 14 characteristics is called the TDI (Total Degree of Influence). The TDI is then multiplied by 0.01 and added to the constant 0.65 to calculate the VAF:

$$\text{VAF} = (\text{TDI} \times 0.01) + 0.65.$$

The final but also optional step in the current IFPUG FP method is to calculate the *adjusted FPs.* To do so, the adjusted FP is then calculated by multiplying the unadjusted FPs with the VAF:

$$\text{Adjusted FP} = \text{unadjusted FP} \times \text{VAF}.$$

Since the 14 GSCs are estimation parameters based on the nonfunctional user requirements, and not part of functional size measurement, only the unadjusted FPs can be considered ISO/IEC-conformant as a functional size measure. *The two steps from the unadjusted FPs to the adjusted FPs take the functional size measurement (unadjusted FP) in the direction of software estimation by considering influences of the nonfunctional requirements in system development.*

In the following sections, each of the IFPUG five function types is described.

Classification of Logical Files

Internal Logical Files (ILF) and External Logical Files (EIF) must be distinguished and counted. The main difference between an ILF and an EIF is that

an ILF is maintained by the application being counted, whereas the EIF is maintained by an external application. The technical term "maintain" is defined as an elementary process that changes the data in the entity (including processes whereby data on the file is modified through processes that create data, update or inserting data, change or otherwise modify data, or delete data). Theoretically, all five manipulations must be possible.

The most important consideration is that the entities (logical files) are regarded from the *user view*. Counting FPs after the design phase or later in the project (e.g., after implementation) can leave the Function Point counting practitioner with difficulties to view everything from the user perspective. The only advice is to remember this restriction as often as possible. A technical perspective (as opposed to the user perspective) can obscure the proper viewpoint and result in an over or under count. For example, an application may physically store data about customers across multiple database files, whereas from the user perspective it is one logical file (entity). This should be counted as one ILF.

A prerequisite to accurate Function Point counting is a *logical data model*, not a physical one. The entities of the logical data model are used for counting and as such the Function Point count will disregard supraentities, IT-technical data elements or implementation specific files, group elements, and filler fields.

The EIFs are external interface files (persistent logical entities) as identified from the requirements. These are logical files (entities) maintained by other *application*s and only referenced by the application being counted. Thus, an EIF is an ILF of another application that is simply read or referenced by this application or one can say it is a logical reuse file.

The complexity of internal and external logical files depends on two dimensions:

- The number of data element types (DET)
- The number of record element types (RET).

IFPUG defines these as follows:

DET: A DET is a unique user recognizable, nonrecursive field (in an ILF or EIF).

RET: A RET is a user recognizable logical subgroup of data elements within an ILF or EIF.

Standalone entities are counted (with the exception of hard-coded/non-maintained data and code tables. See the current IFPUG CPM available from http://www.IFPUG.org for full counting rules and exclusions to what is counted) and the number of fields. When a logical entity contains at least one field, then

a RET is counted. Key fields are counted only once no matter on how many RETs they are contained.

After determination of the RETs and DETs on a persistent logical entity ("file"), the complexity (low, average, or high) is determined using a complexity matrix (see Table 12.4).

The relative complexity is then translated into unadjusted function points according to the following table (Table 12.5).

The example in Fig. 12.12 shows a logical data model of a salary system.

The example in Fig. 12.2 shows 2 RET (since the indicator is functionally required by different restrictions from law in Germany) and 7 DET. Thus, the file is evaluated as low complexity (Table 12.4) and would be equal to seven unadjusted FP if it is an ILF or five unadjusted FP if it is an EIF (Table 12.5). The higher Function Point count for ILFs as compared to EIFs considers that the file is maintained by the application being counted. Note that this means that there will also be at least one data maintenance EI for that ILF present in the application.

New users of the IFPUG method often have difficulties to distinguish between ILFs and EIFs. A rule of thumb is to count an ILF if data are stored and maintained (and are not part of the exclusions as outlined above), and an EIF when data are only retrieved or extracted or referenced from an entity maintained within another application boundary.

One additional piece of advice to determine if the requirement for the file is a physical (i.e., specific to the technical development language or implementation used) or a logical requirement is to consider whether the requirement would disappear if it was implemented differently. For example, if there is a file that contains a copy of information that is maintained by another application, is extracted from that application, imported to the application being

Table 12.4. Complexity of IFPUG data functions: ILF and EIF

RETs/DETs	1–19 DETs	20–50 DETs	>50 DETs
1 RET	Low	Low	Average
2–5 RETs	Low	Average	High
>5 RETs	Average	High	High

Table 12.5. Unadjusted Function Points based on logical file complexity

Complexity	ILF	EIF
	Number of Unadjusted FP	Number of Unadjusted FP
Low	7	5
Average	10	7
High	15	10

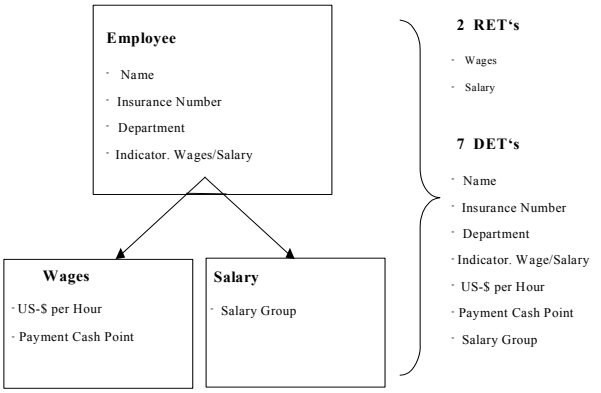

Fig. 12.12. Internal Logical File complexity example

counted, and named with an application specific name, then the question would be "If we had perfect technology (i.e., considering only the user requirements) would we still need to make a copy of the data within our application?" If the answer is "no, we could simply read it from the other application," then we know that the file is an implementation-specific requirement, and the file is simply the physical implementation to read the EIF from the other application. However, if the answer is "Yes, the owner application changes the data all the time, and our application needs a snapshot point in time view of the other application's data", then we know that the requirement is a functional, logical user requirement and the file would be counted as an ILF.

Classification of Transactions

Transactional functions are External Input (EI), External Output (EO) and External Inquiry (EQ), and are defined by IFPUG as follows:

EI: An EI is an elementary process that processes data or control information that comes from outside the application's boundary. The *primary intent* of an EI is to maintain one or more ILFs and/or to alter the behavior of the system. Counted are all elementary input processes having unique processing logic.

EO: An EO is an elementary process that sends data or control information outside the application's boundary. The *primary intent* of an EO is to present information to a user through processing logic other than, or in addition to, the retrieval of data or control information. The processing logic must contain at least one mathematical formula (calculation), create derived data, maintain one or more ILFs, or alter the behavior of the system.

EQ: An EQ is an elementary process that sends data or control information outside the application's boundary. The *primary intent* of an EQ is to present

information to a user through the retrieval of data or control information from an ILF or EIF, and in addition, the processing logic contains no mathematical formulae or calculations, creates no derived data, does not maintain an ILF, and does not alter the behavior of the system.

As such, if an elementary process has the primary intent of sending data external to the application boundary, it typically will be a binary choice between an EO or EQ.

Typical examples for transactions are, for example, the following:

EI: Add a new employee

EO: Online or printed reports with calculated data (can also be contained in an export file)

EQ: Online data is input to retrieve and display employee data without any other processing

The example in Fig. 12.13 shows a dialogue for maintenance of an electronic address book with 3 EIs and 1 EQ.

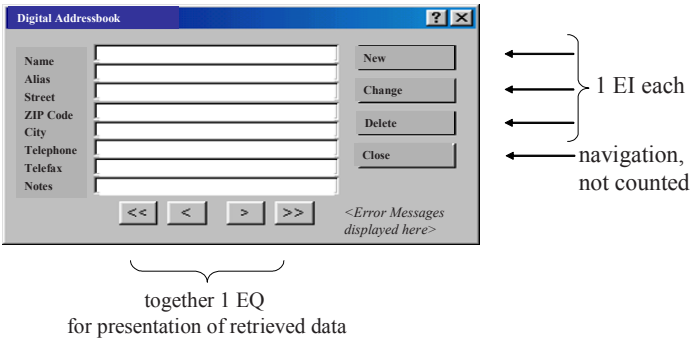

Fig. 12.13. Transactions example

Before counting the unadjusted Function Points, the complexity of each of the transactional functions has to be determined. The complexity of a transaction depends on two dimensions:

- The number of data elements (DET, Data Element Types)
- The number of referenced files (FTR, File Type Referenced).

The number of DETs is determined as the number of data element types that cross the application boundary (in plus out minus duplicate fields that cross both in and out). Counted are the fields used by the transaction plus 1 DET for the ability to specify the function to be performed (e.g., “New” command) plus 1 DET for any error and/or confirmation messages and/or confirm that processing should continue, which are provided as part of the function (regardless of

how many are present, it is 1 DET for the total error/confirmation/continuation messages or functionality there may be).

The number of FTRs is simply the number of external and internal logical files required to process the transaction.

The EI "New" in Fig. 12.13 for adding a new address has, for example, 10 DETs (8 for the data fields shown on the dialog from Name through to Notes, plus 1 DET for the function initiator button New plus 1 DET for the display of error message(s)) and 1 FTR (only a single Internal Logical File is needed to create a new entry). The *complexity matrix for EIs* (see Table 12.6) classifies this EI as low.

Table 12.6. Complexity of EIs

FTRs/DETs	1–4 DETs	5–15 DETs	>15 DETs
0–1 FTR	Low	Low	Average
2–3 FTRs	Low	Average	High
>3 FTRs	Average	High	High

Regarding the complexity of EQs, and EOs, one has to consider that either function may consist of an input part as well as an output part. If a DET is included on both the input (question) and output (response) side, it is counted only once. Therefore, the DETs of both parts are added together, but only the ones that are distinct. The same concept holds for FTRs, where if a FTR is accessed both on the input and output sides of an EQ function, it is counted only once.

The applicable *complexity matrix for EOs and EQs* is presented in Table 12.7.

Table 12.7. Complexity of EOs and EQs

FTRs/DETs	1–5 DETs	6–19 DETs	>19 DETs
0–1 FTR	Low	Low	Average
2–3 FTRs	Low	Average	High
>3 FTRs	Average	High	High

The example data retrieval (at the bottom of Fig. 12.13) function has a primary intent to display information to a user. It retrieves data from a logical file, and the elementary process does NOT involve calculations, derive data, update any ILFs, or alter the behavior of the system. It therefore is an EQ.

The EQ has 3 DETs on the input side (the data field "Name" plus 1 DET for the selection button that identifies the function as a query plus 1 DET for any error messages that can occur) and 1 FTR. The output part has 8 DETs (the data field Name plus the other 7 displayed DETs) and 1 FTR. To determine the complexity of the EQ, use 10 DETs (the 3 DET on the input side + 8 DET on the output side – 1 DET, because the Name field is on both sides), and 1 FTR

(the same logical file is used on both sides). The resultant complexity is low according to Table 12.7.

The Function Points to be counted for the transactions can be derived again using the appropriate column as shown in Table 12.8. The EQ we just counted from Fig. 12.13 is worth three unadjusted FP.

Table 12.8 Unadjusted Function Points of transactions

Complexity	EI	EO	EQ
Low	3	4	3
Average	4	5	4
High	6	7	6

12.6.4 Step 4: Calculate the VAF after Determining the 14 GSCs

After counting unadjusted Function Points, the *VAF* has to be determined. It is calculated in a formula using the sum of *the values 14 GSCs:*

1. Data Communications
2. Distributed Data Processing
3. Performance
4. Heavily Used Configuration
5. Transaction Rate
6. Online Data Entry
7. End-User Efficiency
8. Online Update
9. Complex Processing
10. Reusability
11. Installation Ease
12. Operational Ease
13. Multiple Sites
14. Facilitate Change.

The *Degree of Influence (DI)* of each of these characteristics is rated on a scale from 0 (no influence) to 5 (strong influence). There exists a set of exacting definitions in the IFPUG CPM for determining the DI for each of the 14 GSCs (see Chap. 15, *IFPUG Function Point Counting Rules*). The DI's of the 14 GSCs are added together, and the sum is *called Total Degree of Influence (TDI)*. From this the VAF is calculated with the formula

$$\text{VAF} = (\text{TDI} \times 0.01) + 0.65.$$

This leads to the result that the VAF ranges from 0.65 to 1.35; thus adjusts the unadjusted Function Point count by up to ±35%. A typical VAF (e.g., in the IT department of an international insurance company in Germany) is for

host applications to range from about 1.0–1.1. Experiences of other organizations confirm that VAFs between 0.95 and 1.1 are typical in Europe and elsewhere in the world.

The 14 GSCs correlate strongly with the six categories outlined in ISO/IEC 9126 *Quality Attributes* that play an important part in a quality assurance plan (see also chapter *Estimation Fundamentals: ISO 9126 Quality Attributes and IFPUG GSCs* in this book).

Note that the ISO/IEC 9126 standard is slowly being replaced by the SQUARE series of ISO/IEC standards that expand and further define "Quality Metrics" for software and systems.

12.6.5 Step 5: Calculate the Adjusted FPs

As can be seen from Fig. 12.9 (see *step 1: Define the Type of Count*) the following *three types of count are distinguished:*

1. New development
2. Enhancement
3. Application.

According to the type of count, the Function Points are calculated using specific (and different) formulae as described below.

Function Points for new development projects: A new development project adds functionality to the software application. Further functionality can evolve if existing data must be converted and integrated in the new system (migrations). The adjusted Function Points of a new development project are calculated using the VAF:

$$DFP = (UFP + CFP)VAF,$$

where DFP is the development Function Points, adjusted; UFP is the unadjusted Function Points; CFP is the Function Points from conversions (migrations), which are functions specifically required by users (e.g., user requested conversion reports comparing the results of the existing vs. the new cutover payroll system being installed). These are user-specified and requested reports that are of essence during the development project, but are never put into the production software for ongoing use. (For this reason, the conversion functionality is NOT counted in the base or installed application Function Point count.); and VAF is the Value Adjustment Factor of the application.

Function Points for enhancement projects: An enhancement project changes the functionality of an existing application. The following cases can occur (often all four together):

- New functionality is added
- Existing functionality is changed

- Existing functionality is deleted
- Conversion (migration) functionality is required.

Since the GSCs always pertain to the entire application, they must be evaluated both before and after an enhancement project. Two VAFs are distinguished when calculating the FP for enhancement projects: VAFA and VAFB. The Function Points of an enhancement project are calculated with the following formula:

$$EFP = [(ADD + CHGA + CFP)VAFA] + (DEL \times VAFB),$$

where EFP is the enhancement Function points, adjusted; ADD is the added functionality, new; CHGA is the unadjusted FPs for change of functionality after enhancement; CFP is the unadjusted Function Points for conversion functionality; VAFA is the VAF of application after enhancement project; DEL is the unadjusted FPs for functionality deleted; and VAFB is the VAF of application before enhancement project.

Examples for enhancement of functionality may be as follows:

- A batch transfer for exchange of data with another application is obsolete (deletion of functionality)
- The user demands additional reports from the application (addition of new functionality)
- An already existing report should show additional data elements (change of existing functionality).

Function points of an application: In this case it has to be determined if the application is delivered the first time (initialization of new development) or if an existing application is enhanced (the enhancement project updates the application size). In both cases, when the count is for a project, there may occur conversion (migrations) functionality. Conversion functionality does not change the size of the applications. Hence, the FPs of an application after the completion of a new development project are calculated as follows:

$$AFP = ADD \times VAF,$$

with AFP the application FPs after new development (adjusted), ADD the added functionality of the new development (unadjusted), and VAF the Value Adjustment Factor.

In the situation of the update of an existing application by an enhancement project, the Function Points are calculated according to the following formula:

$$AFP = [(UFPB + ADD + CHGA) – (CHGB + DEL)]VAFA,$$

with AFP the application FPs after new development (adjusted), UFPB the unadjusted FPs before enhancement, ADD the added functionality of the new

development (unadjusted), CHGA the unadjusted FPs for change of functionality after changing it, CHGB the unadjusted FPs for change of functionality before changing it, DEL the unadjusted FPs for deletion of existing functionality, and VAFA the VAF of application after enhancement.

Maintenance projects: Here it has explicitly to be stated that a pure maintenance project does not alter the functionality of an application (i.e., Maintenance projects typically are equal to zero function points). However, if the maintenance project DOES alter the functionality, then it is really an enhancement project according to IFPUG terminology, regardless of what the business might use to classify the project.

Note that this also occurs in the opposite manner: if the business classifies a project as an enhancement, but there is no alteration of any logical functionality in the project, then the project, according to IFPUG FP terminology, is actually a maintenance project and would warrant a Function Point count.

12.6.6 Step 6: Document the Count

This step is not part of the IFPUG method, however it is one of two final steps recommended by the authors - even if the optional adjustment factor steps 4 and 5 are not done. The first Function Point count of a new development project succeeds or fails along with the planning of the measurement. Hence, the right people often have to meet and allocate enough time to review the necessary (requirements) documentation. The final Function Point count of a new development project after delivery occurs at the project postmortem to measure the actual delivered functionality. *This means revisiting and often updating the first Function Point count.*

If the final documentation is complete and structured according to the requirements, then it becomes a trivial matter to update the final delivery Function Point count, and the effort to perform it is minimized.

Persons who neglect to adequately document their Function Point counts could be considered by some to diminish the value (and auditability) of the counting and measurement process itself.

The documentation of a Function Point count should at a minimum comprise the following information:

- The type of count
- Name of the project or application (as applicable)
- Date of the count and name of the counter and participants
- Indication of whether it is a first or final (delivered) count (if the count is for a project)

- Counting practice release used to count (e.g., IFPUG 4.2)
- List of the documentation used for the count (e.g., requirements document version n.n dated dd/mm/yy, object diagram dated dd/mm/yy)
- The system boundary (description and/or diagrams)
- The logical files and transactions
- The elementary processes counted
- A description of any processes or functions excluded from the count (e.g., duplicate functions, menus, or files required for implementation reasons)
- The VAF and the values of the 14 GSCc
- The unadjusted and adjusted FPs
- Assumptions and decisions that had an influence on the count
- Project description and identifying attributes (e.g., platform, development language(s), team size, and any situations that occurred during the project such as changes in direction, delays, changes of management, canceled functionality, etc.).

The process should be at least formal enough that there are usable documents available for subsequent reporting. Thus, at least some forms should be used that enable structured documentation of the aforementioned items. Furthermore, it must be communicated ("publicly stated within the organization") which forms are to be completed to document the FP counts and who is responsible for their completion. In Appendix A of this book there is an example checklist that can be used as a general form to document IFPUG Function Point counts. The form can be tailored for use with other functional size measurement methods.

Furthermore, the same rigor and discipline should be used with these FP measurement activities as is used in accounting, bookkeeping, and controlling departments. When quality assurance of a Function Point count is done by a competence center or by a Certified Function Point Specialist (CFPS), the documentation should be structured in such a manner that any other experienced Function Point counter could understand and get an overview of the Function Point count in the shortest time possible.

Many organizations use an automated tool to document their counts, for example, the FP repository tool: Function Point Workbench™ (FPW) – see the chapter *Tools for Estimation* in this book for details. Tools such as these deliver reports in a structured way (hierarchy diagrams and hierarchy trees) that can easily be prepared for web presentation as well as in other formats. Practical experience of the authors attests that the effort to count enhancement projects and to update the application baseline is minimized when historic counts are available in a structured form as provided by a tool such as FPW™.

The graphical documentation of the application boundary can often be done with graphical software as, for example, MS PowerPoint or Visio, and in the

absence of automated diagrams or tools, even a manual diagram with the boundary depicted can be scanned and attached to the Function Point count automated files (e.g., MS Excel spreadsheet).

Besides it can be recommended to document for each project a log-book (alike a ship's log) with following additional (to the before mentioned) information of the count:

Who did what at which time?
At which time in the project was the count done?
Which special aspects are valid for the project and how is it characterized in the project portfolio?
Which suggestions and decisions were used and for which reasons?
How was the process of the count/estimation performed?
What are the next measures and when are they to be done and by whom?
Which documentation was used for counting?
A cross-reference between physical fields and logical functions.

This logbook is a standard text software document and can be added (as well as the system boundary diagram) to the count documentation in the FPW™ or other FP repository software. In Appendix A of this book we have included an example of such a logbook.

Using the processes described here and tools (checklists and forms as well as software), the organization gains clear, well-structured, and standardized documentation of all Function Point counts. *Enhancement projects* can thus proceed from precise knowledge of the existing application, and can reuse many of the documents as a basis for subsequent enhancement count(s). In addition, it becomes an easy task to verify Function Point counts going forward.

12.6.7 Step 7: Quality Assurance of the Count by the Competence Center

A Function Point count should be reviewed before its final release by a third person or a competence center. In this way, a quality report can show formal or content-related contradictions or weaknesses in the process or in the Function Point counts. For this quality assurance (QA) step, the following three topics can be examined with a QA checklist:

1. Prerequisites
2. Process
3. Documentation.

The first topic checks if the prerequisites for the count were adequate. This requires that the Function Point counters were trained and whether they

were provided with adequate information to gain enough knowledge about the logical functionality of the application and/or project.

The second topic examines the formal process of the Function Point count and, by using random checks ensures that the graphical diagrams and other documentation is consistent with the resultant FP lists of transaction and files (maybe documented in a tool).

The final topic is a check that all necessary information about the Function Point count has been adequately documented.

The results of the checklist can be documented in a short report. Together, the checklist and the report become the quality report. In Appendix A of this book is a sample checklist that can be used to perform the quality assurance of a Function Point count.

12.7 The Process to Implement the IFPUG Function Point Counting Method

The process to implement IFPUG Function Point Counting in an organization is similar to that of introducing estimation as outlined in the chapter *The Implementation of Estimation*. Similar tasks and prerequisite steps must be considered and dealt with before the measurement process becomes an organizational habit and becomes part of the way of doing business. In addition, some of the effort for the implementation of a formal estimation can be transferred to the implementation of the Function Point counting processes.

Günter Büren reports on a project sponsored by the European Community whereby 113 person days of effort were required to implement Function Point counting and an estimating process in a small consultancy.

As a rule of thumb, one can say that an experienced Function Point counter is able to count between 300 and 1,000 Function Points a day. The higher rate of counting can surely be reached if all relevant (and up-to-date) documentation is at hand, and if the count is done with automated tool support. The Function Point counting effort could actually end up to be as much as triple to this if project documentation is not at hand, is incomplete, differs from the implemented application, is not available at all, and if there is no tool support.

On the other hand, the effort for Function Point counts for large IT projects can be in the range of several person days. Professor Dumke states that the software development work of 10 person years can rarely be Function Point counted in a single day.

Note that the minimum time needed to perform a well-documented and detailed Function Point count rarely is less than half a day for the following reasons:

1. It takes time to understand what is involved in the project or application functionality (no matter how big it may be).
2. It takes time to explain the process of Function Point counting to project participants.
3. It takes time to assemble and gain even a high-level appreciation for the needed count documentation and what has been assembled for the count.
4. It takes time to perform the count (even if it is small).
5. It takes time to document and record the information for the Function Point count.

All together, it typically takes at least half a day to be able to do all of these tasks.

Critical success factors for the implementation of IFPUG Function Point Counting in an organization include the following:

- Proper planning of the process to introduce and embed the prerequisite tasks needed to perform Function Point counts in the organization (information gathering, training and participation of management, counters, project team members, and a competency center staff knowing about what Function Point counting can and cannot do; and development and documentation of a Function Point counting process manual and organization specific counting conventions).
- A comparable (stable) development environment where Function Point counting is intended to be applied.
- Realistic expectations about FP based measurement and estimation.
- Committed (and visible) management support.
- An understanding that measurement is a necessary prerequisite for estimation, planning, management, controlling, and improvement of the software development tasks.
- Automated support for measurement and recording of project effort.
- The planning and resource allocation of the necessary effort to learn and become proficient in Function Point counting.
- The readiness to give insight (and feedback) to the processes needed and into the development of necessary documentation.
- Acceptance of the need for control of the processes of measurement and estimation.
- Training of the staff and gaining of experiences in a competence center.

12.8 The Limitations of the IFPUG Function Point Counting Method

The FPM is naturally criticized by users of the SLOC-based metrics, but also by proponents and inventors of alternative methods (such as Mark II, COSMIC). Some of the arguments are politically motivated; however, there remain weaknesses in the IFPUG method (and in other functional size measurement methods).

No matter what methodology is used, the most important consideration is consistency in the application of the method, adequate training of the involved staff, and appropriate usage of an applicable (and calibrated) estimation method.

All investigations so far have led to the result that functional size measurement is the most effective and reliable means of measuring software size that can be used effectively in the early phases of the software development life cycle.

Practically, the question is often raised about the *precision of estimation methods* based on functional size measurement. Shigeru Nishiyama of Japan performed a study of five new development projects in 1999, which were counted by two Function Point Counters (called fpA and fpB) and the results were analyzed thoroughly. His regression analysis resulted in

$$\text{Count by fpB} = 0.97\text{fpA} + 4.01, \quad \text{with } R^2 = 0.999.$$

The negligible difference of 3% between the two counters resulted from different interpretations of vague descriptions in the requirements documents, and from intersections in the declaration of EIs, EOs, and EQs in the IFPUG CPM. See the table *Not defined cases* in the chapter *IFPUG Function Point Counting Rules* in this book.

By design, Function Points do not correlate with every aspect of software development, but they were never intended to do so. FP cannot measure the customer contentedness nor can they be used to measure individual productivity. Pam Morris of Australia documented a list of situations for which FP are not applicable:

- In the area of software maintenance:
  - Defect correction
  - Table changes
  - Perfective and corrective maintenance
  - Production systems support and control
  - Response behavior of the system
  - Security and access control
- Consultancy and ad-hoc support
- Project progress and implementation.

It has been often quoted that if all one has is a hammer, then everything looks like a nail (the analogy to FP measurement is that if you only have FP as a measurement, then everything appears to be Function Point countable). Conversely, a good toolkit contains a combination of tools each suitable to perform particular tasks, such as a screwdriver for screws, a hammer for nails, and a level to hang pictures evenly. Similarly, FP must be balanced by other measures to adequately manage the software development environment.

12.9 Management Summary

The IFPUG FPM is a method to measure the (functional) size of an application (piece of software) from the user view.

As an ISO/IEC conformant Functional Size Measurement method, the IFPUG FPM measures the functionality in software delivered to the user as required by the user, and quantified by following the IFPUG CPM set of counting rules.

The functional size measure is independent of the nonfunctional requirements, including the technology used for implementation, since the technological aspects of the software development are not part of the functional size.

Function Points are derived from the logical (or functional) user requirements concept, and the person counting the Function Points will learn a lot about the functional requirements of the software during the process of evaluating the functional size.

At this point it may be worthwhile to note that there are at least five ISO/IEC conformant functional size measurement methods.

The IFPUG FPM can easily be learned and understood and applied to a variety of software.

It has been observed through first hand experience that when the Function Point Analysis is done with the involvement of the end users, they are motivated to better teamwork and more committed engagement. In addition, we have observed that the overall user satisfaction with the project increases. This is a key project success factor according to the Standish Group's annual CHAOS Report.

FP size at various points in the development life cycle is used for measurement of requirements creep (scope management).

FP methodology supports reuse in IT development by early and standardized quantification of business cases in the requirements definition phase, for contracting, for project-estimation, for test case identification, for enhancements, and for documentation.

The FPM methodology is independent of the development environment as well as the skills or attributes of developers.

Function Points can facilitate comparison of prices from suppliers and to evaluate cost ratios for software under contract.

ISBSG publications also discuss how function points can be used as part of price comparisons (see http://www.isbsg.org). The metric price per FP can contribute to decisions about whether to build-or-buy.

Another interesting use of Function Points is to directly measure and manage software development performed under contract.

The effort for planning, performing, and documentation of Function Point counts can be justified as long as FP are used appropriately together with other measures.

Capers Jones estimates the effort to implement a fully-functioning measurement and analysis program to be maximally 3% of the cost of a project – not much when you consider the savings that better requirements and accurate estimates can provide.

The primary application areas of the IFPUG FPM are in estimation of new software development and enhancement.

FPs can be used for risk assessment.

The COSMIC Method, presented in the chapter about variants of the FPM, claims to address internal processing more concretely, and the reader is directed to select the most appropriate functional size measurement method (amongst the five ISO/IEC conformant methods) to meet their specific needs.

Chris Kemerer from the Massachusetts Institute of Technology (MIT) showed in a research study comparing 15 software projects that the FPM could be used for various types of software beyond management information systems or commercial applications.

Estimating methods based on the FPM for sizing software fulfills the major prerequisite requirements of a method for estimation of effort.

The goals of the IFPUG function point analysis method are to measure small units in order to support flexible comparisons and early deviations from plan.

The optimum time for a first Function Point count is the end of the requirements analysis.

It does not make sense that a Function Point count is performed only once during project progress because throughout the project there evolves new information such as scope changes, clarifications to requirements (as well as requirements creep).

We recommend (consistent with the practices at IBM) to revisit the original Function Point count for any updated information (and changes) at the end of each phase of the software project. This practice also supports the tracking of requirements scope creep and scope management principles.

The counting of Function Points is ideally considered to be a part of the project documentation, reviews, project controlling, and releases at the end of each project phase.

Revisiting the Function Point count at different times as the project progresses enables early adjustments (and corrections) to the resultant effort estimates, and thus it increases the precision and approximation of the actual effort.

When an effort estimate is required at an earlier stage than at the end of the requirements phase, we recommend the development of one or more Function Point prognosis methods.

Function Point counts as well as FP estimates should be performed by project leaders' or project team members' knowledgeable about the functionality together with support of the competence center.

There are three types of Function Point counts, the first two specific to IT projects: New development, Enhancement, Application.

A new development project is the first build of an application. Thus, all delivered functionality is considered to be added.

An enhancement project can add functionality, as well as change or delete functionality.

At project postmortem of an enhancement project, the Function Points of the enhanced application must also be updated based on what has been added, changed, and deleted.

The IFPUG FPM distinguishes between the size of a software project (Counting scope) and an application.

Principally, the application boundary must be defined from the user view.

Since the application boundary is critical to the determination of the application functionality, it is important for it to be documented clearly. This includes the description of assumptions used to locate the boundary.

Since the application boundary is critical to the determination of the application functionality, it is important for it to be documented clearly. This includes the description of assumptions used to locate the boundary.

Practically, this documentation (typically including system diagrams) can easily be reused in (or as) architecture diagrams in the application atlas of the organization.

The two steps from the unadjusted FPs to the adjusted FPs take the functional size measurement (unadjusted FP) in the direction of software estimation by considering influences of the nonfunctional requirements in system development.

ILF and EIF must be distinguished and counted.

The EIFs are external interface files (persistent logical entities) as identified from the requirements.

The complexity of internal and external logical files depends on two dimensions: the number of DET and the number of RET.

New users of the IFPUG method often have difficulties to distinguish between ILFs and EIFs. A rule of thumb is to count an ILF if data are stored and maintained (and are not part of the exclusions as outlined above), and an EIF when data are only retrieved or extracted or referenced.

Transactional functions are EI, EO, and EQ.

The complexity of a transaction depends on two dimensions: the number of data elements (DET) and the number of referenced files (FTR).

After counting unadjusted Function Points, the VAF has to be determined. It is calculated in a formula using the sum of the values 14 GSCs.
The 14 GSCs correlate strongly with the 12 ISO/IEC 9126 Quality Attributes that are an important part of a quality assurance plan.

The first Function Point count of a new development project succeeds or fails along with the planning of the measurement. Hence, the right people often have to meet and allocate enough time to review the necessary (requirements) documentation.

The final Function Point count of a new development project after delivery occurs at the project postmortem to measure the actual delivered functionality. This means revisiting and often updating the first Function Point count.

Persons who neglect to adequately document their Function Point counts could be considered by some to diminish the value (and auditability) of the counting and measurement process itself.

The process should be at least formal enough that there are usable documents available for subsequent reporting.

Furthermore, the same rigor and discipline should be used with these FP measurement activities as is used in accounting, bookkeeping, and controlling departments.

Besides that it can be recommended to document for each project a logbook (alike a ship's log).

A Function Point count should be reviewed before its final release by a third person or a competence center.

As a rule of thumb, one can say that an experienced Function Point counter is able to count between 300 and 1,000 Function Points a day.

The Function Point counting effort could actually end up to be as much as triple to this if project documentation is not at hand, is incomplete, differs from the implemented application, not available at all, and if there is no tool support.

On the other hand, the effort for Function Point counts for large IT projects can be in the range of several person days.

No matter what methodology is used, the most important consideration is consistency in the application of the method, adequate training of the involved staff, and appropriate usage of an applicable (and calibrated) estimation method.

All investigations so far have led to the result that functional size measurement is the most effective and reliable means of measuring software size that can be used effectively in the early phases of the software development life cycle.

By design, Function Points do not correlate with every aspect of software development, but they were never intended to do so. FP cannot measure the customer contentedness nor can they be used to measure individual productivity.