4.1.2 Function Count

Measuring software size in terms of lines of code is analogous to measuring a car stereo by the number of resistors, capacitors and integrated circuits involved in its production. The number of components is useful in predicting the number of assembly line staff needed, but it does not say anything about the functions available in the finished stereo. When dealing with customers, the manufacturer talks in terms of functions available (*e.g.*, digital tuning) and not in terms of components (*e.g.*, integrated circuits).

Alan Albrecht while working for IBM, recognised the problem in size measurement in the 1970s, and developed a technique (which he called Function Point Analysis), which appeared to be a solution to the size measurement problem [ALBR79, ALBR83]. It measures functionality from the users point of view, that is, on the basis of what the user requests and receives in return. Therefore, it deals with the functionality being delivered, and not with the lines of code, source modules, files, etc. Measuring size in this way has the advantage that size measure is independent of the technology used to deliver the functions. In other words, two identical counting systems, one written in 4 GL and the other in assembler, would have the same function count. This makes sense to the user, because the object is to buy an accounting system, not lines of assembler and it makes sense to the IT department, because they can measure the performance differences between the assembler and 4GL environments [STEP95].

Function point measures functionality from the users point of view, that is, on the basis of what the user requests and receives in return from the system. The principle of Albrecht's function point analysis (FPA) is that a system is decomposed into functional units

- Inputs : information entering the system.
- Outputs : information leaving the system.
- Enquiries : requests for instant access to information.
- Internal logical files : information held within the system.
- External interface files : Information held by other systems that is used by the system being analyzed.

The FPA functional untis are shown in Fig. 4.3.

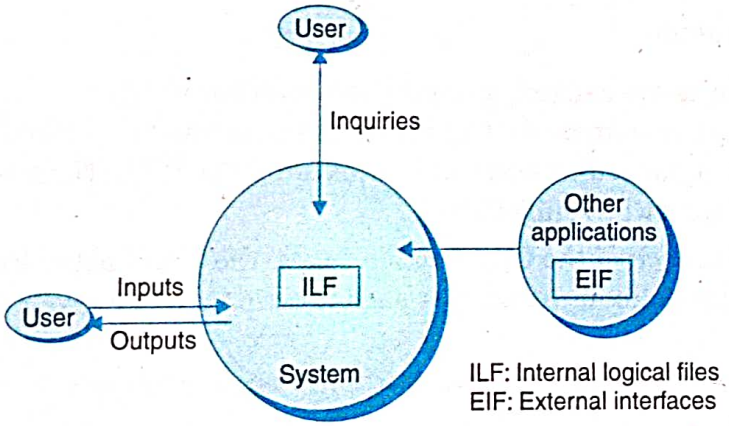

Fig. 4.3: FPAs functional units

The five functional units are divided in two categories:

(*i*) **Data function types**

- *Internal Logical files (ILF):* A user identifiable group of logically related data or control information maintained within the system.
- *External Interface files (EIF):* A user identifiable group of logically related data or control information referenced by the system, but maintained within another system. This means that EIF counted for one system, may be an ILF in another system.

(ii) **Transactional function types**

- *External Input (EI):* An EI processes data or control information that comes from outside the system. The EI is an elementary process, which is the smallest unit of activity that is meaningful to the end user in the business.
- *External Output (EO):* An EO is an elementary process that generates data or control information to be sent outside the system.
- *External Inquiry (EQ):* An EQ is an elementary process that is made up of an input-output combination that results in data retrieval.

Special features

- Function point approach is independent of the language, tools, or methodologies used for implementation; *i.e.*, they do not take into consideration programming languages, data base management systems, processing hardware or any other data base technology.
- Function points can be estimated from requirement specification or design specifications, thus making it possible to estimate development effort in early phases of development.
- Function points are directly linked to the statement of requirements; any change of requirements can easily be followed by a re-estimate [INCE89].
- Function points are based on the system user's external view of the system, non-technical users of the software system have a better understanding of what function points are measuring.

This method resolves many of the inconsistencies that arise when using lines of code as a software size measure [MATS94].

Counting function points

The five functional units are ranked according to their complexity *i.e.*, Low, Average, or High, using a set of prescriptive standards. Organisations that use FP methods develop criteria for determining whether a particular entry is Low, Average or High. Nontheless, the determination of complexity is somewhat subjective.

After classifying each of the five function types, the Unadjusted Function Points (UFP) are calculated using predefined weights for each function type as given in Table 4.1.

Table 4.1: Functional units with weighting factors

Functional units	*Weighting factors*		
	Low	*Average*	*High*
External Inputs (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiries (EQ)	3	4	6
Internal logical files (ILF)	7	10	15
External Interface files (EIF)	5	7	10

Table 4.2: UFP calculation table

Functional units	*Count complexity*		*Complexity totals*	*Functional unit totals*
External Inputs (EIs)		Low × 3 =		
		Average × 4 =		
		High × 6 =		
External Outputs (EOs)		Low × 4 =		
		Averge × 5 =		
		High × 7 =		
External Inquiries (EQs)		Low × 3 =		
		Average × 4 =		
		High × 6 =		
Internal logical files (ILFs)		Low × 7 =		
		Average × 10 =		
		High × 15 =		
External Interface files (EIFs)		Low × 5 =		
		Average × 7 =		
		High × 10 =		
Total Unadjusted Function Point Count				

The weighting factors are identified (as per Table 4.1) for all functional units and multiplied with the functional units accordingly. The procedure for the calculation of Unadjusted Function Point (UFP) is given in Table 4.2.

The procedure for the calculation of UFP in mathematical form is given below:

$$\text{UFP} = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

where i indicates the row and j indicates the column of Table 4.1.

w_{ij} : It is the entry of the i^{th} row and j^{th} column of the Table 4.1.

Z_{ij} : It is the count of the number of functional units of Type i that have been classified as having the complexity corresponding to column j.

Organisations that use function point methods develop a criterion for determining whether a particular entry is Low, Average or High. Nonetheless, the determination of complexity is somewhat subjective.

The final number of function points is arrived at by multiplying the UFP by an adjustment factor that is determined by considering 14 aspects of processing complexity which are given in Table 4.3. This adjustment factor allows the UFP count to be modified by at most ± 35%. The final adjusted FP count is obtained by using the following relationship.

$$FP = UFP * CAF$$

Where CAF is complexity adjustment factor and is equal to $[0.65 + 0.01 \times \Sigma F_i]$. The F_i $(i = 1$ to $14)$ are the degrees of influence and are based on responses to questions noted in Table 4.3.

Table 4.3: Computing function points

Rate each factor on a scale of 0 to 5.

0	1	2	3	4	5
No Influence	*Incidental*	*Moderate*	*Average*	*Significant*	*Essential*

Number of factors considered (F_i)

1. Does the system require reliable backup and recovery?
2. Is data communication required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing heavily utilized operational environment?
6. Does the system require on line data entry?
7. Does the on line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on line?
9. Is the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organisations?
14. Is the application designed to facilitate change and ease of use by the user?

Uses of function points

The collection of function point data has two primary motivations. One is the desire by managers to monitor levels of productivity, for example, number of function points achieved per work hour expended. From this perspective, the manager is not concerned with when the function point counts are made, but only that the function points accurately describe the size of the final software project. In this instance, function points have an advantage over LOC in that they provide a more objective measure of software size by which to assess productivity.

Another use of function points is in the estimation of software development cost. There are only a few studies that address this issue, though it is arguably the most important potential use of function point data.

Functions points may compute the following important metrics:

Productivity	=	FP/persons-months
Quality	=	Defects/FP
Cost	=	Rupees /FP
Documentation	=	Pages of documentation per FP

These metrics are controversial and are not universally acceptable. There are standards issued by the International Function Point User Group (IFPUG, covering the Albrecht method) and the United Kingdom Function Point User Group (UFPGU, covering the MK11 method). An ISO standard for function point methods is also being developed.

The function point method continues to be refined. So if we intend to use it we should obtain copies of the latest international function point user group (IFPUG) guidelines and standards. IFPUG is the fastest growing non-profit software metrics user group in the world with an annul growth rate of 30%. Today it has grown to over 800 corporate members and over 1500 individual members in more than 50 countries.