

Early Function Points :

a new estimation method for software projects

Roberto Meli (CFPS) - D.P.O. Srl

ESCOM97 - Berlin - May 1997

Abstract

Function Point Analysis is a software measurement technique which is gaining wider diffusion all over the world even if some experts pointed out various problems affecting it. The present paper describes a new technique for Function Points estimation (Early and Extended Function Points Method) with the objective to solve some of these problems. The use of the proposed estimation technique allows:

- To anticipate, in the life-cycle of a software project, the determination of the number of Function Points (FP) to be developed or maintained;
- To highly reduce time and cost in the formulation of the measurement usually required by a detailed evaluation;
- To keep into due consideration the aspects related to software reuse;
- To add the algorithmic complexity among the elements affecting the technical sizing, in order to extend the applicability of the method to "technological" softwares (e.g., telecommunication, real time software);
- To better evaluate a software based on a distributed architecture (with different general and productivity characteristics in its various components), by grouping the FP into homogeneous environments.

The application of the method allows to obtain two different measurements: Early Function Points (EFP) and Extended Function Points (XFP).

Early Function Points are not a measurement alternative to FP IFPUG 4.0, but only a faster estimation of the same. Therefore, factors of conversion between the two measures are not required. The number of EFP approximates the number of FP IFPUG 4.0 when both calculated on the same user requirements. EFP are not affected by aspects relative to reuse, algorithmic complexity and grouping since these aspects are currently neglected by IFPUG standards.

As a contrary, XFP derive from EFP after the application of the three correction factors mentioned above (reuse, algorithmic complexity and grouping). XFP are not compatible with FP IFPUG 4.0, but they can be used for more accurate productivity evaluations.

The paper includes the results of a real estimation experience occurred in a major Italian company comparing the results of counts calculated according to IFPUG 4.0 standards with the EFP estimates, hereafter exposed, showing their strong correlation.

The technique object of the present paper was implemented in a business product called Sfera supporting both IFPUG 4.0 standard and Early and Extended Function Points method, hereafter briefly described.

Key words

Software Engineering; Software Cost Estimation; Metrics; Early Function Points; Function Points.

About the author

Roberto Meli is Director of DPO Srl. In the last 20 years he has been dealing with Project Management and Software Metrics. He writes articles for technical magazines and international conferences. In 1996 he obtained the IFPUG certification as an expert in Function Points Analysis. He is consultant and lecturer in training courses on the above-mentioned subjects for some major Italian Companies. He created and developed the Early and Extended Function Points method, managing the implementation of the Sfera product. Currently, he is coordinator of the GUFPI (Gruppo Utenti Function Points Italia) Counting Practices Committee.

e-mail: roberto.meli@iol.it

mail: via Flaminia, 217 - 00196 Roma (Italia)

Introduction

In the relatively short story of computer technology applied to business management, only recently software practitioners have realised that, in order to obtain satisfactory and repeatable results in software projects development and maintenance, it is necessary to engineer in the proper way the production processes. At the beginning, software experts had to concentrate on the definition of development process models and working techniques for the analysis of data, functions, etc. Once these models and paradigms became stable, mature and sufficiently diffuse, it was possible to proceed to a more practical study of the software metrics field, in particular of those metrics applicable to project management: in fact, we know that without measurements it is not possible to understand and, as a consequence, even to control any working process. Among the current measurement techniques we find the Function Points method, established by an international organisation: IFPUG (International Function Points Users Group). This measurement tool is deemed to be particularly appropriate for the business information systems software domain.

Function Points are gaining an increasing international consensus since they meet at least three fundamental needs experienced by organisations: to measure software on a "business value" basis from the user's point of view; to obtain measures as independent as possible from the technological contexts in which the same applications are developed and used; and, finally, to use the same measures in order to forecast effort and duration of a development project or of an enhancement/maintenance project.

The first need is linked to the fact that a measurement of this type works to facilitate the dialogue between the technician and the user or between the supplier and the client. It allows them to concentrate on the central aspects of the supply, i.e., on "what" the user can do with a given software (functions and data), instead than on "how" to realise it from a technical point of view. The second need is linked to the necessity to compare on a technological basis different solutions, aiming anyway at the same business objectives. The purpose is to obtain indirect indications of the productivity relative to a given development environment or to the quality of a specific final product. The third need is of a managing nature and is linked to the necessity to plan the resources involved in the resolution of a software project, with a higher level of certainty, rather than mere intuition.

Another important advantage to be taken into account in considering an FP measurement, is that it is possible to determine its value at an early stage, i.e. when detailed functional users' requirements of business application are evident and available. Unfortunately, to the purpose of Project Management this anticipation is not sufficient, since the level of detail necessary to the application of IFPUG standard counting rules implies that a big portion of the project is already realised (Functional Specification takes 15 to 40 % of the total work effort). Otherwise, it would be impossible to identify external inputs, external outputs, external inquiries, internal logical files and external interface files without incurring in relevant evaluation errors. In fact, if we consider the importance of an estimation with respect to the management of the project, we come to face a curious and maybe paradoxical phenomenon: measurement is very useful when we do not have enough elements to obtain it (Feasibility Studies), but when we can identify it with absolute accuracy (just before the final product is ready) it is not necessary any more (at least for effort predicting purposes).

Here comes the need to have at our disposal a value estimation method of FP measurement already usable after an accurate Feasibility Study. This has to roughly define what lately is going to be subject to a deeper detailed analysis which can allow the real FP counting. In fact, among the experts it is more and more common that the word "counting" is related to the use of IFPUG standard rules for the identification of FP values, while the word "estimation" is used for all those alternative forecasting techniques of the same FP values.

To simplify what said before, it must be considered that a skilled FP analyst would be able to assign an FP measure to an application simply observing the specifications and entrusting his intuition and experience. The final result of this process cannot be called "counting", even if, depending on the case and the person, it can be very close to the number obtainable through the application of the same specific detail rules. The technique herewith proposed has as a main purpose that of providing the analyst with an instrument to improve the estimation rate through a method alternative to intuition and therefore more objective, verifiable and evolutionary.

Why do Function Points work?

Before explaining this new evaluation method, it is useful to spend a few words on the validity of the Function Points technique, sometimes object to criticisms for its assumed lack of rigour. What follows does not want to be a formal validation of FP technique, rather a call to pragmatism.

The fundamental thesis against FP technique holds that the counting procedure and result features do not allow its validation with respect to the formal framework of the measurement theory. Of course, this point is true and cannot be objected.

The real problem is that the software itself does not show simple and evident characteristics useful to the measurability of its own technical dimensions. A software is not a liquid or a solid body whose "volume" and "mass" can be detected and measured. Up to date, we do not even have a shared definition of what software is. How is it possible that a number attributed to something of such a vanishing nature can have its own formal characteristics? Actually, formal theories applied to software very often resulted in poor usefulness to the business world. How many organisations do use formal techniques and their relative theorems in order to prove the rightness of each program? In practice, all we do is to find all possible "test cases", hoping that the actual use of the software created will not present situations too different from those conceived when tested. The software that nowadays rules the world was created through too little industrial and automated processes to be subject to the rules of theoretical or applied mathematics.

Also in the case of the "size" of a software it is very difficult to agree upon a definition of the attribute to be measured; therefore, we have to proceed using a vague image of what we think to be a software. For example, as far as Lines of Code are concerned, are we sure that we know what they do measure, once we agreed upon what a Line of Code is? Do they measure programmer's attitude, tools used to program, functionalities for the user, megabytes stored in a computer memory or what else ?

Software applications do not have a visible "volume" or "size" allowing them to be arbitrarily dismantled and analysed, according to their own constituent elements. Taking one element off the program can affect the proper functioning of the whole system. Therefore, applications cannot be put on an ordinal scale with respect to these attributes since they are not homogeneous entities, and the meaning itself of

"volume" or "size" is not clear at all. Not only the mathematical operation of the difference between two applications does not have any sense, but also their addition, as the result of the fusion of the two applications could be a software incompatible with itself, non-functioning and therefore of no use. Thus, we have to accept the idea of imperfection and lack of rigour in anything related to a product with such a high level of indefiniteness, such as a software today. At the moment, in this field the engineering approach is better than the mathematical one.

Anyway, in practice all this does not bear any particular negative consequences for the following reasons:

- In contrast with what happens for other disciplines, software measure does not affect the activity of the product construction as much as it affects its management and evaluation. In other words, while it is necessary to measure the mixture of elements in order to obtain a concrete of good quality and build a solid and reliable bridge, the same is not valid for a software, since in order to create a program it is not necessary to measure its dimensions. As a matter of fact, in almost all business environments, metrics activities tend to be considered as a management overhead, with no immediate return for those who created the program. Instead, measurement is necessary in case we want to determine the dimension of a working group on the basis of the activities to be developed, or in the evaluation of the fault density of the application, and so on. Thus, in general it is especially related to the purposes regarding the management of the production or business process. These purposes are quite far from mathematical rigour.
- Even if it is impossible to agree upon an exact definition of a software and its "dimensions", it is still possible to detect "pseudo"-measures of some use from the practical point of view.
- The essential point is that, given a body of rules to associate a measure to an application, the measure has to be repeatable and independent from the performer.
- If the number obtained from the application of the rules mentioned above results empirically correlated to some software phenomena we want to forecast and control (quantity of operations and/or data the user can utilize; effort, duration and cost of development, etc.), why should we refuse it only because the measure is not formally validated?

Concluding this brief and certainly not exhaustive analysis, we want to assert that Function Point measure sufficiently meets the requirements just mentioned. Therefore, this is a case of "pseudo-measure" and it is useful to utilize it as long as new and more advanced metrics tools are available, even if they still seem very far because of the quite primordial state of software engineering as a scientific discipline.

Instead, what we can definitively agree upon is that the FP method can be improved under many aspects. For example: the Value Adjustment Factor (VAF), which depends on 14 general system characteristics, takes independence from technologies and productivity off the measure, leaving it less "functional". It would be better to abandon the factor, and then take those and other general characteristics into account only during the identification of the development productivity phase. In fact, it does not seem correct to increase or decrease up to 35% of FP working on elements different from External Input, External Output, External Inquiry, Internal Logical File and External Interface File, the basic bricks of measurement. Surely, Unadjusted Function Points more accurately measure the different options the user has at his disposal in dealing with functions and data, while VAF could be a qualification index of the functions themselves.

Origin and objectives of the Early Function Points technique

The main reasons that convinced us to look for and define a new FP evaluation method are the following:

- The managing and business need to anticipate as soon as possible the results of the measurement, in order to solve the foresaid evaluation paradox.
- The need to dramatically decrease time and cost required in a detailed evaluation, both for projects to be developed and for existing applications.
- The desire to experiment extensions which, once recognised, could produce improvements in the standard method, such as:
 - To keep into due consideration the aspects connected to software reuse;
 - To add algorithmic complexity among the elements for the evaluation of the technical sizing, in order to extend the applicability of the method to *technological* software;
 - To evaluate in a better way a software based on a distributed architecture with

different general and productivity characteristics in its various components.

The first and second objectives were attained through the introduction of the Early Function Points (EFP) concept, the third one through the introduction of the Extended Function Points concept (XFP).

Early Function Points

For a given application, EFP constitute an estimate of the corresponding Function Points IFPUG 4.0, obtained by the application of a different body of rules.

The method is based upon the identification of functionalities and data, provided by the software, at different levels of detail. In fact, we can reach a deeper knowledge of a specific application branch because it has already been realised more than once in a similar way, while we know little or nothing at all about another branch, which is completely innovative. In the first case, we can proceed to a deeper level of detail, while in the second one we have to limit us to a rougher level of attribution. Of course, the degree of uncertainty in the estimation (expressed as the distance between the minimum estimation and the maximum one) is wider when we stop at a superficial level of the classification and will be the weighted sum of the single components' uncertainty level.

A **functionality** of a software product constitutes a whole of actions performed by the computer system, enabling any kind of users to attain a business objective (of an operational, general or strategic level), through computer technologies. Functionalities can be classified as: Macrofunction, Function, Microfunction and Functional Primitive. A statistics table, where to each kind of functionality corresponds a FP number (min, avg, max), can allow the assignment of the right value of EFP for each functionality identified in a given application.

A **functional primitive** is the smallest correlated whole of actions, performed with the help of a computer system with autonomy and significant characteristics. It allows the user to attain a unitary business objective of operational level. It is not possible to proceed to further useful decompositions of a functional primitive, from the expert/user point of view.

There are three types of Functional Primitives: **Primitive Input, Primitive Output and Primitive Inquiry**. In conceptual terms, they correspond to the elementary processes of the standard Function Points Analysis, i.e. External Input, External Output and External Inquiry. These are functionalities such as

the introduction of new addresses in a client file, the cancellation of a hotel booking-record, and so on.

The weight in function points of functional primitives is increased in relation with the weight of the correspondent elementary processes in order to take into account the fact that there is a physiological rate of elements impossible to submit to a superficial analysis, what happens at the early stage of the software life cycle (for example, all the implicit inquiries useful to fill in some fields in a screen mask).

The first aggregated functionality of the model is called a **Microfunction** that constitutes the whole of 4 functionalities: create, read, update and delete (CRUD) of a record in a logical file. Usually, these aggregations are associated with the term "management of ...", meaning that any file should be subject to all the described operations so that it would be possible a useful utilisation of the software application.

Instead, a **Function** is an aggregation of a superior level (a set of average functional primitives) which can be assimilated to an operational sub-system of the application in order to allow the attainment of an articulated whole responding to the users' objectives. A function can be small, medium or large according to the fact that it groups an estimated number of functional primitives (see table below).

small	up to 11 functional primitives
medium	from 12 to 18 functional primitives
large	from 19 to 25 functional primitives

Finally, a **Macrofunction** is an aggregation of large dimensions (a set of medium functions) that sometimes can constitute a whole development project by itself. It can be assimilated to a relevant sub-system of the entire Company Information System, such as the general accounting system, the marketing system, the production management system, etc. A macrofunction can also be small, medium or large (see table below).

small	up to 3 functions
medium	from 4 to 7 functions
large	from 8 to 12 functions

The attribution of the right level to a functionality is a very delicate process because of the EFP model sensitivity. Each object we have just been defining is associated to three values in terms of Function Points (min, avg, max) deriving from empiric observations and considering the so-called "good rules" of software engineering. These assignments can be subject to improvement on the basis of data collection activity and statistical analysis for actual projects. The

ultimate end of EFP is to attain the closest approximation to FP IFPUG 4.0 and therefore any kind of improvement has to be considered with favour.

In order to avoid misusings of the EFP model, we suggest the following advice:

1. If in the application documentation for the same functionality there are different levels of detail it is better to use the most detailed one;
2. A functionality subject to classification belongs to an inferior level (detailed), unless there are "crushing" proofs it belongs to the following one.

Besides functionalities, also files should be classified on a five-level scale of general complexity, whose first three levels exactly correspond to those considered in IFPUG's rules, while the other two are for particularly complex macro-files, grouping different logical files of the detailed standard count. For the first three levels it is possible to use the same IFPUG 4.0 complexity tables, based on the number of elementary fields (DET) and given sub-groups (RET), if this information is available. The difference between internal logical files (maintained by the relevant application) and external interface files (maintained by other applications) can be neglected, since is not always clear at a superficial analysis level. In case of files, the EFP number is intermediate to ILF and EIF one in the standard method.

Once the function and data typology and complexity are detected, it should be possible to use the relevant conversion tables in order to determine the value in Unadjusted Early Function Points of the various elements (see appendix). Finally, after UEFP are summed for each functionality and each file, it should be possible to apply the Value Adjustment Factor in order to determine the Adjusted Early Function Points number of the application. A last intervention can be performed to take into account the requirements' variation occurred since the time of an anticipated study up to the definition of Functional Specifications. That is a variation percentage which can increase or decrease the EFP number in order to come to an approximation of the count according to IFPUG standards. This percentage cannot be subjective, but is the result of statistical research performed in the productive environment and is not considered in this paper.

The reliability of the method is directly proportional to the user's capability to "recognise" the components of the application as part of one of the classes described. This capability can sharpen through practice by comparing different counts obtained using

standard rules with those relative to EFP. This kind of sensitivity can be attained at a quick step if it is possible to work on about ten projects with different occasions of comparison, or after proper training. Anyway, the dimensional rules in appendix should reduce decisional uncertainty.

Of course, EFP method is subject to a certain degree of subjectivity, but this is not a counting method (which must be as strict and exact as possible); it is an anticipated estimation method with the role of expanding the decisional abilities of a human estimator.

Extended Function Points

Extended Function Points represent a variation of the EFP method with the addition of the following intermediate steps:

- After functionalities and files classification, it is possible to associate them with a multiplier factor assuming values from 0 to 1 and representing "saving due to reuse", with respect to a complete development of the functionality starting from scratch;
- If the functionality belongs to a basic level (in which case it is a functional primitive) it can be individually corrected by a multiplier factor representing the intrinsic complexity of the algorithms internally used. In order to reduce subjectivity in attribution, we may refer to a classification by Tom De Marco in his work *Controlling Software Projects*;
- Each functionality or file is associated with a group expressing diversified general system characteristics (for example, client and server architecture), each with its own Grouping Value Adjustment Factor (GVAF).

Of course, the EFP number thus determined does not correspond to the FP IFPUG 4.0 number, but in a sense it is a more accurate measure of what the development or enhancement maintenance project has "actually" to develop. In fact, the first variation factor takes into account the fact that the reuse of an existing software (documents, code, specifications, user manuals etc.) probably involves savings which can be quantified using a standard scale of options. The second variation factor takes into account the algorithmic aspects of elementary processes, increasing or decreasing FP of a functionality according to the complexity of the operations performed on data. Also in this case it is possible to use a closed list of options to reduce subjectivity as it is exactly possible for the single general system

characteristic of the standard method. The third extension takes into account the fact that today softwares do not develop any more in a monolithic environment and that FP of a functionality should be corrected by a factor representing the architecture and the development procedures in which it is collocated.

Therefore, the XFP number is not going to be the FP number the user receives (reused functions are anyway functions the user can utilise and thus they have to be properly considered), but it is the number useful to the evaluation of the actual productivity of the project and to the estimation of development efforts and costs. In fact, at this point it is appropriate to notice that the FP method bears an ambiguity element since its birth. We can say it has two parallel souls, each of one justifying this or that kind of choice. They are: to measure the value for the users and to be useful to the evaluation of the productivity development. Although the first objective is officially accepted, the second one is present in some of the technical choices it would be difficult to explain otherwise. For example, why do External Input weigh less than External Output and the same as External Inquiry? Is it because they attribute less value to the users or maybe because with '80s technologies (when FP were born) it was easier to build a data introduction mask than a print report?

It is thus necessary to consider two corrective factors of the functional measure: the first one has to determine what the user receives in terms of functions and data he can utilise (even if he has to reuse what already elaborated); the second one has to be useful to forecast the work needed taking into account reuse *discount*. This is particularly important for the quality of a benchmarking data base which can release productivity data "altered" by intensive reuse phenomena, together with software data newly elaborated. In a contractual client-supplier relationship, in general the former is not able to know the productive procedures of the latter and thus has to base himself on the non-discounted FP. The supplier, who is able to "reuse" many times, can show an apparent productivity greater than the average, turning out to be particularly advantageous, or he can increase his operative margins. Therefore, reuse can be considered (and certainly it is) a competition factor. On the other hand, to forecast an effort starting from a dimension in FP without taking into account reuse, means an overestimation of the necessary resources with a high risk of waste. In fact, if we consider the project as a gas occupying

the space offered by the container, we could even not realise the fact that we are wasting resources, as these are anyway absorbed by the project itself.

Enclosed herewith there is a classification table of the measures referred to in this paper.

Case study

The following table shows an actual case of the application of the two methods (IFPUG 4.0 and EFP) on 5 business applications, part of an Information System of a major Italian Company. The methods were applied on the basis of the same documentation (even if more or less detailed) in order to eliminate the "scope creep" influence, i.e. the requirement variation which would have altered the comparison. Counts were performed by IFPUG certified staff.

The results can be considered encouraging and widely respondent to a commonly accepted criterium for the validation of a forecasting model (75% of the observations show an error within a range of +/-25%) although in this case the number of observations is too low to be taken into account for statistical considerations. It even seems that summary estimates (for which 1 element was analysed out of 20 for the detailed evaluation) produce better and more equally distributed results.

Relation with other estimation methods

Among anticipated estimation methods, it is quite common to resort to procedures that, starting from the identification of the logical files present in the software project, consider elementary functions as proportional to the number of the files themselves (create, update, delete and display for each logical file). As a matter of fact, this is a particular way to use the method proposed in this paper. If we only wanted to count the logical files and the relevant micro-functions, we would obtained the same results. Anyway, EFP allow a deeper analysis accuracy, since they can extend or reduce the number of foreseen functions for each file with respect to the basic assumption.

With respect to Capers Jones's Feature Point which, besides transactions and files, also considers number and type of the algorithms present in the application (but only at a global level), XFP allow the association of *each elementary process* with the prevalent type of algorithm and its relative degree of complexity. Therefore, this is a more detailed system of evaluation that only needs an inter-company standardisation, as it was already done by IFPUG for the General System Characteristics and the Value Adjustment Factor.

In our opinion, Function Points Mark II constitute a metrics with a conceptual model incompatible with FP IFPUG one and therefore it is impossible to compare them. In fact, the basic measured objects are of a different nature. In case of FP MK II, the elementary process derives from Input, Processing and Output logic according to which each elementary process has an input, processing and output component and refers to logical files. According to FP IFPUG, an elementary process must have one single priority: either input or output. Even Inquiry, with both natures, does not allow any possible algorithm but the mere data retrieval. Since the elementary objects on which the measure focuses are different, it is impossible to compare their results and it is senseless to work to find an equivalence between the two different scales.

The Sfera software product

D.P.O. has realised a software tool supporting both the standard Function Point Analysis (IFPUG 4.0) and the Early and Extended Function Point method. Sfera automatises the most mechanical steps of the two techniques, allowing to focus the attention on the actual important aspects of the counting or estimation processes. One of the advantages of using a software tool is that of consenting the standardization of the variables and parametres needed through the use of a closed list of options. In this way it is possible to reduce the relative subjectivity of the estimation process. This tool is currently in use in some major Italian Companies.

Case study results

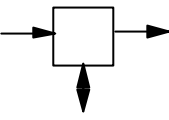
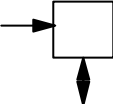
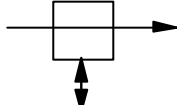
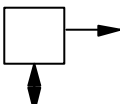
Application	UFP IFPUG	Unadjusted Early Function Points					
		detailed		intermediate		summary	
		value	error	value	error	value	error
System 1	516	465	-11%	452	-14%	474	-9%
System 2	621	653	5%	564	-10%	652	5%
System 3	173	189	8%	185	6%	169	-2%
System 4	230	260	12%	248	7%	253	9%
System 5	595	697	15%	645	8%	608	2%
Total	2135	2264	6%	2094	-2%	2156	1%

Essential references:

- Geert Poels - Why F.P. do not work: in search of new software measurement strategies - Guide Share Europe - August, 1996
- IFPUG - Function Point Counting Practices Manual, Release 4.0 - Westerville - Ohio, 1994
- Jones, C. - Feature Points - IFPUG Fall 1988 Conference, Montréal Québec, 1988
- Symons - Function Points Analysis : Difficulties and Improvements - IEEE Transactions Software Engineering - Vol. SE-14 n.1, 1988
- Tom De Marco - Controlling Software Projects : management, measurement & estimation - Englewood Cliffs, NJ - Prentice Hall, 1982

Comparative table of the measures mentioned in this paper

Name	Type	Use of GVAF/VAF	Extentions	Notes
Unadjusted Function Points (UFP)	Not weighted	No	No	Produced by ' <i>counting</i> ' activities It is a "pure" functional (pseudo)measurement of the technical size of a software product It is determinable only by using detailed specifications
Adjusted Function Points (AFP)	Weighted	Yes (VAF)	No	Produced by ' <i>counting</i> ' activities It is a functional (pseudo)measurement of the technical size of a software product "contaminated" by non-functional factors It is determinable only by using detailed specifications
Unadjusted Early Function Points (UEFP)	Not weighted	No	No	Produced by ' <i>estimating</i> ' activities It is a "pure" functional (pseudo)measurement of the technical size of a software product It is determinable also by using summary specifications
Adjusted Early Function Points (AEFP)	Weighted	Yes (GVAF)	No	Produced by ' <i>estimating</i> ' activities It is a functional (pseudo)measurement of the technical size of a software product "contaminated" by non-functional factors It is determinable also by using summary specifications
Extended Function Points (XFP)	Weighted	Yes (GVAF)	Yes : reuse complexity multiple groups	Produced by ' <i>estimating</i> ' activities It is a conventional value linked to the actual productivity environment It is not representative of the functionalities released to the user

Elements table				
		Estimated UEFP per element		
		min.	avg.	max.
Logical File				
simple		5	6	7
average		8	9	10
complex		13	14	15
very complex		14	17	21
multiple		15	21	27
Macro-function				
small		151	215	280
medium		302	431	560
large		603	861	1119
Function				
small		45	56	67
medium		73	91	109
large		106	133	160
Micro-function				
MF		16	18	20
Functional Primitive				
PI	primitive input 	4	5	7
PQ	primitive inquiry 	4	5	7
PO	primitive output 	5	6	8

Functional Complexity		
c1	generic process	1.0
c2	data direction	0.3
c3	amalgamation	0.6
c4	separation	0.6
c5	simple calculations	0.7
c6	editing	0.8
c7	storage management	1.0
c8	text manipulation	1.2
c9	data retrieval	1.0
c10	data verification	1.0
c11	processes synchronization	1.5
c12	complex graphical display	1.8
c13	complex computation	2.0
c14	device management	2.5