Volume 4, Issue 12, December 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Survey on Function Point Estimation Methods

Saoud Sarwar
Ph.D Research Scholar
Deptt. of Computer Science & Engineering,
Bhagwant University, Ajmer, Rajasthan, India

Naveen Verma
Professor
Deptt. of Computer Science & Engineering,
Bhagwant University, Ajmer, Rajasthan, India

Abstract- A function point is a unit of measurement to express the amount of business functionality an information system (as a product) provides to a user. Function points measure software size. The appearance of the Function Point technique has allowed the ICT community to increase significantly the practice of software measurement, with respect to the use of the traditional "Lines of Code approach". A FP count, however, requires a complete and detailed level of descriptive documentation, like the Functional Specifications of the software system under measurement, to be performed. There are at least two situations in which having an estimation method, compatible but alternative to the standard rules for FP, could be decisive. The first case occurs when the development or enhancement project is in such an early phase that it is simply not possible to perform a FP count according to the IFPUG standards (i.e. in the Feasibility Study). The second case occurs when an evaluation of the existing software asset is needed, but the necessary documentation or the required time and resources to perform a detailed FP calculation are not available. Based on these and other analogous situations, the demand of methods for estimating - not counting - Function Points has risen from the organizations involved in software business. The technical literature offers several estimation methods that can be examined and compared. This paper presents, therefore, the characteristics of some outstanding methods (Early Function Points, ILF Models, and Backfiring) and a general benchmarking model, useful for the evaluation of any additional method, as well as encourage in the field of three point cost estimation technique.

Keywords: Software measurement, estimation, benchmarking, size, function point, three point cost estimation.

I. A SHORT INTRODUCTION TO FUNCTION POINT ANALYSIS

FP (Function Points) is the most widespread functional type metrics which is suitable for quantifying a software application. It is based on 5 user identifiable logical "functions" which are divided into 2 data function types and 3 transactional function types (Table 1). For a given software application, each of these elements is quantified and weighted, counting its characteristic elements, like file references or logical fields.

Table 1. Complexity weights with corresponding number of UFP.

	Low	Average	High
ILF (Internal Logical File)	7	10	15
EIF (External Interface File)	5	7	10
EI (External Input)	3	4	6
EO (External Output)	4	5	7
EQ (External inQuiry)	3	4	6

The resulting numbers (Unadjusted FP) are grouped into Added, Changed, or Deleted functions sets, and combined with the Value Adjustment Factor (VAF) to obtain the final number of FP. A distinct final formula is used for each count type: Application, Development Project, or Enhancement Project.

II. COUNTING OR ESTIMATING?

FP Analysis is generally considered an early and efficient software sizing technique but, considering the standard IFPUG Counting Practices, it implies the availability of a complete and detailed set of descriptive documentation of the user functional requirements for any software application to be measured.

There are at least two situations in which having an estimation method, compatible but alternative to the standard rules for FP, could be decisive. The first case occurs when the software development or enhancement project is in such an early phase that it is simply not possible to perform a FP count according to the IFPUG standards (i.e. in the Feasibility Study). A standard count, in fact, requires the identification of elements that are not always identifiable at the beginning of a project, when, however, the need for effort, time and cost forecasts based on size evaluation is surely stronger than at the end of the Functional Specification Phase. The second case occurs when an evaluation of the existing software size asset is needed, but the necessary documentation or the required time and resources to perform a detailed FP calculation are not available.

Besides the correct standard measurement process, that should always be carried when it is possible, various estimating methods have been proposed to respond to the need of evaluating the size of a software project as soon or with the smallest commitment of resources as possible: *"estimating"* means using less time and effort in obtaining an approximate value of FP.

The advantages of an estimation technique are obviously counterbalanced by the unavoidable minor accuracy of the FP exploitations, but this aspect is usually considered by the users of the methods themselves.

Therefore, we should always strongly distinguish between different terms and concepts: *"counting FP"* should mean measuring software size through the use of the standard IFPUG practices, while *"estimating FP"* should denote an approximate evaluation of the same size through other different means.

III. A GENERAL BENCHMARKING MODEL FOR FP ESTIMATING METHODS

In order to compare different FP estimation methods, we need to define a general model for classifying the estimation methods themselves and a correspondent *benchmarking model*

A. Classification of FP Estimation Methods

An estimation method may be represented as an input-processing-output system where the input variables are, essentially, information on the software application to be "sized" whereas the output variable is the functional size expressed in FP. Both the input and the intermediate variables are measured through the use of consistent metricses. The estimation methods may be classified in two main categories: Direct Estimation and Derived Estimation. The former groups all the methods based on analogical reasoning and intuition where the FP result is achieved directly without the formalization of a step by step analytical process. The latter groups all the well defined algorithmic or structured methods based on theoretical or statistical models.

Any estimation method may be applied using bottom-up or top-down approaches to the decomposition structure of the software to be estimated.

Bottom-Up and Top-Down Estimation Bottom-up estimation begins with the lowest level components, and provides an estimate for each of them. The bottom-up approach combines low-level estimates into higher-level estimates. Top-down estimation begins with the overall system. Estimates for the component parts are calculated as relative portions of the full estimate.

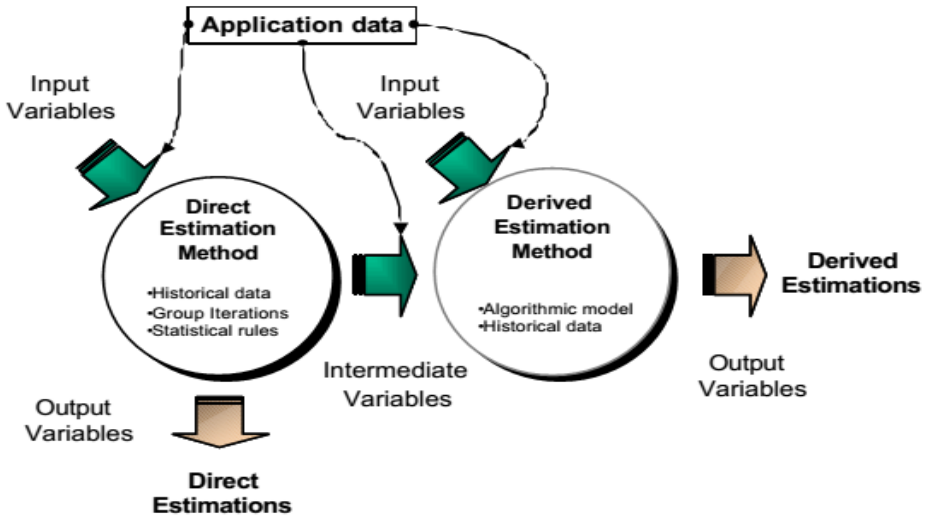

Fig 1. Generic conditions about different methods

We may classify the different estimation methods into the following categories; Table 2 reports some generic considerations about these categories.

1) Direct Estimation Methods: These are also known as Expert Opinion Methods. They involve consulting one or more experts, who will provide a direct guess of the FP estimation required, based mainly on past experience, intuition and unconscious processes. Quite frequently an expert knows what the right value is but is unable to explain why it is so. The mental process is similar to an iceberg, the biggest part of the reasoning being under the surface of the conscience. A direct estimation may be improved by the use of some techniques like Delphi method (or its Shang variant) or the Analogy. The former is an iterative group technique that allows the participants to better the individual estimates through

the collaboration of different points of view in the estimation effort. The latter involve reasoning by analogy with one or more completed software applications to relate their actual sizes to an estimate of the size of a similar new application. Analogy can be made between high or low level components of the software application.

2) Derived Estimation Methods: These methods, also known as Algorithmic Model Methods, provide one or more transformation algorithms which produce a software size estimate as a function of a number of variables which relate to some software attributes. Generally, these methods are correlated to a decomposition process. By decomposing an application into its major functions, estimation can be performed in a stepwise fashion. The difference between a direct estimation and a derived estimation is mainly that in the second case the measurements or the estimations are not made directly on the FP values but on different software attributes which are supposed to be correlated with the FP values themselves.

3) Some General Considerations on the Estimation Methods:

Table 2. General comparison of software size estimation methods

Method Category	**Strengths**	**Weaknesses**
Direct Guess	Suitable for atypical projects	No better than participants
(individual or Delphi)	May result in very accurate and shared estimates	It may be difficult to justify the results
Simple & Structured Analogy	Based on representative experience	Depends on the availability of historical data
	Tends to consider all the software factors	Depends on the expertise of the estimator(s)
Algorithmic Model	Objective, repeatable, analyzable formulae	Not suitable for atypical projects
	May be calibrated to historical data	Calibrated to past, not future
	May be easily taught to and used by beginners	Disregards components not included in the model

B. Definition of a General Benchmarking Model

A General Benchmarking model should take into account a set of quality *attributes* or *parameters,* linked to the object and the aim of comparison. Each parameter, for each alternative, must be assigned the appropriate *value*, on an absolute scale.

Depending on the circumstances and the actual needs of the organizations, the same estimation method can be evaluated as optimal in some cases, and not so good in other cases. So, the benchmarking model should let us express both objective and subjective variables, i.e. both the actual performances of the method with respect to the considered parameters and the relative weights of the parameters themselves. One appropriate technique is that of the Weighted Choice Matrix:

Table 3: Weight Choice Matrix

Weighted Choice Matrix							
		Alternatives					
		Method A		***Method B***		***Method …***	
Parameters	***Weight***	***Score***	***WxS***	***Score***	***WxS***	***Score***	***WxS***
Parameter 1							

Parameter 2							
Parameter …							
Totals			***Tot.A***		***Tot.B***		***Tot…***

The *Weight* column represents the different priorities of the organization which is evaluating the estimation methods. A vote from 1 to 10 may be used consistently. The weights may be assigned using a Delphi technique, since they should be shared at the organization level. The *Score* column contains the evaluation of the performance of a particular alternative with respect to the parameters chosen. Again, a vote from 1 to 10 may be used consistently. The *Weighted Score* column is the result of the multiplication
between the priorities (weights) and the performances (scores). The best alternative is the one which shows the highest sum of the weighted scores in the appropriate column. An example should clear this point. Let us consider an organization which is mainly involved in responding to public competitions, offering software development services without the possibility of changing the economical conditions of supply after the assignment. In this case the organization will be very interested to methods which are the most accurate as possible even if they are more costly (time or effort) than others. So, the weight for the parameter "accuracy" will be significantly higher than the weight for the parameter "cheapness". An internal ICT department which main problem is to assign a pre-allocated budget to different projects with the possibility of moving resources from one project to another during the development phases, will probably be more interested in a "quick and dirty" estimation method using different priorities in the Weighted Choice Matrix.

C. The Comparison Parameters

All or some of the following criteria should be included in the benchmarking model in order to evaluate the goodness of a software size estimation method.

1) Method Quality Parameteres:

Definition	Is the method described appropriately in a technical document? Does the method define exactly which results it can provide, and which ones it excludes? Are input variables precisely defined? Is it specified what to take into account, and what to ignore in measuring the variables?
Compatibility	Are the method's concepts compatible with the IFPUG framework? Are the same (kinds of) objects investigated?
Scope	Does the method cover all the classes of variables / counts whose values you need to estimate? I.e. Application asset, Development project, Enhancement project, ADD, CHANGED, DELETED functions.
Adaptability	Is it easy to modify, update, calibrate the method itself?
Precocity	How early is it possible to use the method in the Software Life Cycle?
Power	What is the level of detail and the completeness of the information needed for the input variables to produce meaningful results? Is it possible to feed the method with much less details than the standard IFPUG technique?
Cheapness	Is it a low-cost and low-time consuming estimation method?
Parsimony	Does the method avoid the use of highly redundant factors, or factors which make no appreciable contribution to the results?
Robustness	May eventual manipulations of the presentation in the input data invalidate the final results ?
Invariance	Is it hard to "jigger" the method to obtain any result you want?
Reliability	Are both the input data and the results reliable? Is the estimation process repeatable? Is it statistically consistent?
Stability	Do small differences in input variables produce small differences in output estimates?
	Validity Has the method been presented in appropriate technical contexts and validated by independent researches? Are the historical data a valid sample?
Easiness	Are the method's inputs and options easy to understand and specify?
Constructiveness	Can a user tell why the method gives the estimates it does? Does it help the user to understand the job to be done in order to obtain the estimation?
Standardization	Is the method usable by a large class of users in the same way? Does it allow too many variants?

2) Input Variables Quality Parameters

Independence	Are the input variables independent one from each other? Are redundancies avoided?
Technical	Does the method avoid the use of input information which will not be Availability well known at the moment of utilization?
Organizational	May organizational or environmental resistances make it difficult or input data?

Availability	
Objectivity	Does the value to be assigned to the input variable depend on the particular viewpoint or the estimator experience?

3) Output Variables Parameters

Accuracy	Are the estimates close to the actual standard measurement results? Is the estimate equally likely to be above or below the actual result?[1]
Articulation	Does the model easily accommodate the estimation of the different components, as, for example, data groups and transactions, besides the total result?
Confidence	Is it possible to forecast the estimation probable error? The size estimate should be expressed as a range with a maximum and minimum bound, more than a single number.

IV. REVIEW OF FP ESTIMATION METHODS

Let's now review the most common estimation methods. We note here that no method provides special considerations about estimating the VAF (Value of Adjustment Factor), since this value can be easily determined even when few details are available for a standard count; most of the reviewed methods focus on the Unadjusted FP estimation.

A. Direct Estimation Methods

The Expert Opinion approach to estimation may results in very accurate estimates, although it is entirely dependent on the experience of the expert(s). Sometimes, in case of many experts collaborating to the same effort, it may be difficult for the estimates to converge to a unique value. The estimate itself may be influenced by subjective factors, as personal relationships, contractual aspects, and so on.

1) Delhi or Shang Techniques: These are among the most commonly used procedures[2, 3] , by which individual predictions are combined. In brief, they provide iterated cycle of anonymous estimations by each expert, until the estimates converge to an acceptable range. The result is a group estimate arrived at by consensus. The group estimate is typically a better overall estimate than any individual prediction.

*2) Three Point Estimation Technique:*This is a technique to improve direct estimation, when more values are provided by estimators. Given the Minimum, the Most Likely, and the Maximum Value for the size, the estimate is:

$$Est.Size = (Min + 4 \times MostLikely + Max) / 6$$

with standard deviation:

$$s = (Max - Min) / 6$$

3) Simple Analogy Methods: The most common way to apply a Simple Analogy Estimation for software size is to look for a known system in the historical database, that is "similar" (in analogical sense) to the application under estimation. The found implemented system provides a quick estimate of the new project size. Further investigation likely leads to Structured Analogy (see next section)

*4) Structured Analogy Methods:*These are more formal approaches than the Simple Analogy Method. The estimator compares the proposed application to one or more existing applications: s/he typically identifies the type of application, establish an initial prediction, and then refine the prediction within the original range. Passing from a Simple Analogy to a Structured Analogy, differences and similarities are identified and used explicitly in a mathematical way to adjust the estimate. A concept of "distance" among systems may be defined and used to prioritize the choices.

B. Derived Estimation Methods

There are many Algorithmic Model Methods for estimating FP size, mostly because of statistical researches and benchmarking results. Not every method is cited in technical literature, but many of them are widely known in practice.

1) Extrapolative Counts: These approaches assume that we are able to count only one FP component (typically the number of Internal Logical Files) of the application, and derive the rest of the count on a statistical or theoretical basis. All of these models should be carefully analyzed, in order to understand their exact applicability.

For example, the [24] shows a strong correlation (statistically significant) between the number of ILFs and the Unadjusted FP count of the sample of systems, which were of a similar type, mostly batch:

$Est.UFP = \#ILF \times 11.01$ (if Batch System)

$Est.UFP = \#ILF * 14.93$ (if Transactional System, CreateReadUpdateDelete)

Tichenor ILF Model provides also a specific regression for deriving the estimated Adjusted FP:

$Estimated\ FP = 1.0163 \times (UFP \times VAF) \wedge 1.0024$

Additional FP may arise if it's explicitly known that other functions differs from these average ratios to ILF (i.e. the system is not "typical" of the Tichenor's sample):

EI:ILF = 0.33, EO:ILF = 0.39, EQ:ILF = 0.01, EIF:ILF = 0.097

FP Prognosis by **[19]** provides similar ratios, but for a sample of systems that are mostly online:

EI:ILF = 2.66, EO:ILF = 3.14, EQ:ILF = 1.20, EIF:ILF = 0.40

FP Prognosis derives a so-called rule of thumb, from the sum of the quantities of EI and EO (IO):

$FP = 7.3 \times \#IO + 56$

This formula may be used for early estimation when the data component (ILF and EIF) is not investigated. The

correlation is statistically strong, but the sample is very small. The error range is 20%. The researcher from CNV AG reminds that, "of course, a complete FP count at the end of the requirements analysis is obligatory."

Another public data driven approach [16] considers a template of elementary processes for each identified ILF: for a GUI application it is often CRRUDO (Create, Read All - drop down list, Read One - expand one item from the list, Update, Delete, and Output). The model lets assign decimal values to each element, but for sake of easiness we may consider values more according to the IFPUG rules:

Data Group (ILF itself, Low) = 7 UFP

Create, Update, Delete (3 EI, High) = 3 × 6 = 18 UFP

Read All (EQ, Low) = 3 UFP, Read One (EQ, Average) = 4 UFP

Output (EO, Average) = 5 UFP

So, this model sums up 37 UFP for each identified ILF. EIF must be counted apart, if present. Users estimates that this model should take 20% of the counting time. More general templates are in the range of 32-40 UFP per ILF (often 35-38 for GUI).

NESMA (Netherlands Software Metrics Association) [20] proposes the **Indicative FP**, also known as "the Dutch method", calculated from the number of data functions (ILF and EIF), not even weighted for complexity:

Indicative Size (UFP) = 35 × #ILF + 15 × #EIF

The Indicative FP Count is based on the assumption that there will be about three EI (to add, change, and delete information in the ILF), two EO, and one EQ on average for every ILF, and about one EO and one EQ for every EIF.

From the Release 5 of the [6], we can extrapolate a more generic set of ratios (for Developments Projects): on the average over 400 cases, ISBSG found:

ILF 22.3%, EIF 3.8%, EI 37.2%, EO 23.5%, EQ 13.2%

The ISBSG Benchmark also shows that most ILF are rated with Low complexity (i.e. 7 UFP, or more precisely 7.4 average UFP) so we may state:

UFP (only ILF) = 7.4 × #ILF

UFP (total) = UFP (only ILF) / 22 × 100

ISBSG states that a contingency of 20-30% should be added, because of the so-called "implicit" functionalities, not apparent in the early stage, but then included in standard FP counts.

Similar methods arise if we have visibility of other types of functions, not ILF. When we can identify and count all the 5 types, then the method is classified as Average Complexity Estimation.

2) Sampled Counts: Using this method, the IFPUG count of a part of the system is made and from this partial count the rest of the system is estimated. While in the preceding method the whole system is investigated with respect to some FP components (EI, EO, EQ, ILF or EIF), in the Sampled Count a portion of the system is investigate with respect of all the FP components.

3) Average Complexity Estimation: Release 5 of the **ISBSG Benchmark [10,11]** shows the following average function complexity for Development Projects (similar values are found also by CNV AG):

Table 4: Average Fuction complexity for development project

	Average UFP	IFPUG
ILF	7.4	10
EIF	5.5	7
EI	4.3	4
EO	5.4	5
EQ	3.8	4

So, if we identify all the components (EI, EO, EQ, ILF, and EIF), these function types may be assigned a weighted average complexity rating (there is no need to see their internal structure):

Est.UFP = #EI × 4.3 + #EO × 5.4 + #EQ × 3.8 + #ILF × 7.4 + #EIF × 5.5

[24] Proposes a similar formula ("IRS Fast Count"), with slightly different ratings, while **NESMA** [16] simplifies this approach with its "Estimated FP" (generally more accurate than the Indicative FP Estimation):

"Rate the complexity of every data function (ILF, EIF) as Low, and of every transactional function (EI, EO, EQ) as Average" (therefore, using IFPUG rates).

4) Catalogues of Typical Elements: This method requires the creation of "catalogues" of typical elements; one catalogue should contain all the typical elementary processes that can be identified for any project, and their average number of UFP (more precisely, the catalogue should list typical processes, grouped by project domain and other characteristics). **Frallicciardi *et al [4].*** suggest to restrict the vocabulary used to describe the required functionalities, in

order to quicken the transaction identification. In brief, one single, well-defined verb should describe each process.
This method should be associated with Average Complexity Estimation, by ISBSG coefficients.

5) Early Function Points: Early FP technique combines different estimating approaches in order to provide better estimates of a software system size; this method makes use of both analogical and analytical classification of functionalities. In addition, it lets the estimator use different levels of detail for different branches of the system (multilevel approach): the overall global uncertainty level in the estimate (which is a range, i.e. a set of minimum, more likely, and maximum values) is the weighted sum of the individual components' uncertainty levels. The multilevel approach makes it possible to exploit all the knowledge the estimator has on a particular branch of the system being estimated, without asking questions difficult to answer in an early stage or, on the contrary, leaving some detailed information on a portion of the system unused. Finally, the Early FP method provides its estimates through statistically validated tables of values.

The key factors in an Early FP estimate are: Macro functions, Functions, Micro functions, Functional Primitives and Logical Data Groups. In conceptual terms, Functional Primitives correspond to the elementary processes of the standard FP Analysis, i.e. EI, EO and EQ, while Macro functions, Functions, and Micro functions are different aggregation of more than one Functional Primitive at different detail level. Logical Data Groups correspond to standard Logical Files, but without any differentiation (in the present version of the method) between "external" and "internal" data, and with some levels suitable for aggregation of more than one logical file.

Each "object" is assigned a set of FP values (min, avg, max) based on analytical tables, then the values are summed up to provide the UFP estimate (these assignments are subject to improvement on the basis of statistical analysis for actual projects, as from the ISBSG research); as for the most of the other estimation methods, the Adjustment Factor is determined similarly to the standard IFPUG method. Estimates provided by this method may be denoted as "detailed", "intermediate" or "summary", depending of the detail level chosen (or forced) for the early classification of functionalities. Please refer to the cited works in the reference for exact definition, guidelines and numerical indicators of this method.

The reliability of the EFP method is directly proportional to the estimator's ability to "recognize" the components of the system as part of one of the proposed classes. This ability may sharpen through practice by comparing the different counts obtained using standard and Early FP rules. However, the Early FP technique has proved quite effective, providing a response within ± 10% of the real FP value in most real cases, while the savings in time (and costs) can be between 50% and 90%, with respect to corresponding standard counts.

6) Backfiring: This term denotes the derivation of the FP value, for a given software system, from its quantity of LOC (Lines of Code). This approach originated from a research by Capers Jones, who provides a detailed table of language levels: as language levels go up, fewer statements to code one Function Point are required.

Besides implicit difficulties in using LOC (e.g. "what exactly constitutes one LOC", "no standard definition of LOC", "how to estimate LOC early in the development"), we remind the fact that FP is supposed to be a "functional" measure of the size of an application - what to do - whereas LOC is supposed to be a "technical" measure of the size of an application - how to do it. LOC measure something that is conceptually far away from "functional size".

If there was any kind of general and stable relation between FP and LOC than this should mean one of the following:

- LOC is not a "technical" but a "functional" measure
- FP is not a "functional" but a "technical" measure
- "technical" measures are also "functional" measures

We cannot believe to the theoretical validity of backfiring without believing to one of the preceding options. The authors personally don't believe to any one of them.

An example should clarify this point. A system may include 5 different External Inputs which use a common data validation functionality. This functionality weights half the total number of LOC for each EI. A structured programming approach would suggest to write the validation code once, to be "called" as many times as it is needed. In this way the same 5 EI could correspond to X LOC or to 3 *X / 5 LOC depending on the programming style. Actually, LOC is related to implementation software modules whereas FP is related to the logical functionalities that those physical modules aim to implement. The relationship between logical transactions and physical modules is usually "many to many". There's no surprise if backfiring may lead to very high levels of error. It should be considered a very quick, but very rough and risky way to estimate the size of a system. Although a LOC may be easily denoted as new, modified, or deleted, no precise linking is provided between LOC and transactional or data function types.

Other kinds of "backfiring"

Various attempts have been made to link "physical" elements as Pages of Documentation, Screens, Reports, Physical Files, or Entities from Entity-Relationship Diagrams to the FP measure. Mostly, the same considerations as for LOC Backfiring apply.

Nishiyama reports the Takahashi regression formula between numbers of **Screens, Reports and physical Files** (of course for systems with external designed already completed), and the size of the system in FP (the reliability is very high for the analyzed sample, and error margin less than 20% in most cases):

$$Est.FP = 10.99 \times R + 4.52 \times S + 17{,}57 \times F - 510.45$$

Or

$$Est.FP = 12.31 \times R + 6.01 \times S + 8.05 \times F$$

About the relation between **Entities and FP Counts**, some consultants in 1997 reported:

$$FP\ Approximation = \#\ Subject\ Areas \times 8 \times 33 \times 1.25$$

Or

$$FP\ Approximation = \#\ Entity\ Types \times 0.8 \times 33 \times 1.25$$

Where:

each subject area on the average will contain 6 to 10 entity at the end of the project,

33 comes summing 10 (ILF maintained), 12 (3 EI, create, modify, delete, average complexity), 2 (0.5 EI), 4 (1 EQ, view or list), 5 (1 EO, report)

1.25 for considering that at the beginning on the average only the 75% of the effective requirements are known apparently

0.8 in the second formula, since on the average 20% of the entities are grouped together for FP counting purposes.

There is one special kind of backfiring approach which is particularly harmful: the **cost driven backfiring**. According to this practice, firstly an estimation of the total cost of the development or enhancement project is made using any mean. Secondly, adopting a market standard fixed price per FP, the size estimation is made in a backward fashion. In this way the FPs of the application are not those actually linked to the user requirements to be implemented but are those needed to justify an expenses budget to be allocated. This practice, unfortunately, is quite frequently adopted by acquisition departments of commercial companies. The final effect is to transform FP in a currency unit like Dollar or Euro: budgets are directly quoted in FP instead of money.

7) Database Objects Mapping: **Kellen[11]**, from Kallista Inc, states that "in database development, certain objects can be equated with the FP counterparts. Reports can be roughly equated with EO, forms and dialog boxes with EI and tables with ILF. EIF are files that communicate to other applications. EQ are screens which let the user ask for help in the system."

Some hints for counting single data element types from the tables' and forms' field are provided, too. This analogy should be statistically validated at a higher order.

8) Use Cases Mapping: Use cases have become a common method for capturing user requirements. One benefit of UC is that each one encapsulates a set of requirements. This encapsulation lets easily manage and track UC individually and provides a better alternative to prose requirements. In a design review, UC force designers to show how each UC is implemented by the design and which items in the design are not part of any UC. This ensures all requirements are implemented (and no unnecessary work is done). Because UC describe functionality from the user's point of view, they *should* be easily converted to FP. This fact *must be validated* item by item, since the level of "dissection" of the functional transactions and of UC may be not the same. Further research should provide some statistical evidence of this eventual relationship.

9) Objects and Methods Mapping in OO Environment: **Catherwood *et al [2].*** found (with a small sample) some significant correlations between FP and number of objects and methods:

Mean number of objects per "use case FP" is 1.159, □ = 0.045

Mean number of objects per "implemented FP" is 0.380, □ = 0.120

Mean number of methods per "use case FP" is 18.182, □ = 8.444

Mean number of methods per "implemented FP" is 4.955, □ = 0.901

It's relevant the fact that in this case the "estimate" of FP from Use Case documentation yields a very different result from the "Implemented FP" count.

Fetcke *et al[3].* provides a conceptual framework, and a concrete set of rules, for mapping the OO-Jacobson approach into FP Analysis; this approach is only a starting point for standard counts, since it only gives a list of possible elements candidates), which must then be evaluated with IFPUG rules. This, in the authors' opinion, confirms the diversity of conceptual elements between the OO environment and the FP framework. Fetcke *et al.* also cite some different approaches (Whitmire, ASMA, Karner, Gupta), most of which don't respect the compatibility with the standard IFPUG statements.

As afinal consideration we may state that, with the exception of UC, an OO representation is linked to an implementation technology more than to a logical view of the software application: for this reason, the same considerations made for the backfiring techniques hold in this case too.

Other statistical or theoretical research in the OO field is needed, anyway.

V. CONCLUSION

The main conclusion that we can draw is that none of the alternatives is better than the others from all aspects: indeed, the General Benchmarking Model has been set up in order to find the optimal estimation method for each given situation.

We must note that the level of detail of information needed to estimate FPs following the most part of the Derived Estimation Methods is very similar to that one needed by a standard count and this means that the corresponding estimation methods are not particularly powerful. This is not true for the Early Function Point method.

The strengths and weaknesses of the different methods are complementary (particularly the algorithmic model versus expert opinion), so it should be tried the *use of a combination of techniques*, and the *comparison and iteration of the estimates* obtained from each one. In general, an effective combination is the following:

Top-down estimation using the judgment of more than one expert, using analogy estimation where a comparable previous project is available.

Bottom-up estimation using an algorithmic (or advanced) model, with inputs and component-level estimates provided by future performers.

The authors wish also to encourage participating in Three point cost estimation techniques as well as continuous public benchmarking programs, since this action can also improve the specific field examined in this work.

REFERENCES

[1] Bundschuh M. - Function Point Prognosis - FESMA 98 Procs, May, 1998
[2] Jin Y ongqin, Li Jun, Lin Jianming, Chen Qingzhang, "Software Project Cost Estimation Based On Groupware", World Congress on Software Engineering, IEEE, 2009.
[3] Y . F . Li, M. Xie, T. N. Goh, "A Study of Genetic Algorithm for Project Selection for Analogy Based Software Cost Estimation, IEEE, 2007.
[4] Chetan Nagar, Anurag Dixit, "Software efforts and cost estima- tion with sy stematic approach", IJETCIS, ISSN:2079 -8407, V ol.2 No.7, July 2011.
[5] V ahid Khatibi, Dayang N. A. Jawawi, "Software Cost Estimation Methods: A Review", Journal of Emerging Trends in Computing and Information Sciences, ISSN 2079-8407, V olume 2 No.1, pg. no 21-29,2010-11.
[6] Catherwood B., Sood M., AMS - Continued Experiences Measuring OO System Size - ESCOM 97 Procs - May, 1997
[7] Fetcke T., Abran A., Nguyen T.H. - Mapping the OO-Jacobson Approach into Function Point Analysis - TOOLS USA 97 Procs, 1997
[8] Frallicciardi L., Natale D. - Estimating Size Using Function Point Analysis and Analogy - ESCOM 97 Procs, May, 1997
[9] Herrygers J. - Estimating Projects for Accuracy and Continous Improvement - IFPUG Conference, February, 1996
[10] Hill P.R. (ISBSG) - Software Project Estimation, A Workbook for Macro-Estimation of Software Development Effort and Duration - March, 1999 - Chapter 3
[11] IFPUG - Function Point Counting Practices Manual, Rel. 4.0 - January, 1994
[12] IFPUG - Guidelines to Software Measurement, Release 1.1, Sep. 1996 - Chapt. 13
[13] Jones C. - Programming Languages Table, Release 8.2, March 1996 - www.spr.com
[14] Johnson K.. - Software Size Estimation - Dept. Of Computer Science, University of Calgary, January, 1998
[15] Vince Kellen, Kallista, Inc. - Estimating and Tracking Software Projects
[16] Londeix B. - Three Points Techniques in Software Project Estimation - SEER - April, 1997
[17] Longstreet D. - FP Applied to New & Emerging Technologies - ESCOM 98 Seminar, May, 1998
[18] Meli R. - Early and Extended FP: a New Estimation Method for Software Projects - IFPUG Fall Conference - September, 1997
[19] Milne B.J., Luxford K.B.G. (ISBSG) - Worldwide Software Development, The Benchmark, Release 5 - March, 1998 - Chapters 5-6, pp. 23-36
[20] NESMA (Netherlands Software Metrics Users Association), www.nesma.nl
[21] Nishiyama S., NTT - On More Effective Uses of FP Analysis- May, 1998 - FESMA 98 Procs
[22] Santillo L., Meli R. - Early Function Points: Some Practical Experiences of Use - ESCOM- ENCRESS 98 - May, 1998
[23] Tichenor C. - The IRS Development and Application of the Internal Logical File Model to Estimate Function Point Counts - IFPUG Fall Conference, September, 1997
[24] Wyder T. - Capturing Requirements with Use Cases - Software Development Magazine, Features - February, 1996 (www.sdmagazine.com)

