An Evaluation of Function Point Counting Based on Measurement-Oriented Models

Vieri del Bianco
University of Insubria
vieri.delbianco@uninsubria.it

Claudio Gentile
University of Insubria
claudio.gentile@uninsubria.it

Luigi Lavazza
CEFRIEL and
University of Insubria
luigi.lavazza@uninsubria.it

OBJECTIVE: It is well known that Function Point Analysis suffers from several problems. In particular, the measurement criteria and procedure are not defined precisely. Even the object of the measurement is not defined precisely: it is given by whatever set of documents and information representing the user requirements. As a consequence, measurement needs to be performed by an “expert”, who can compensate the lack of precision of the method with the knowledge of common practices and interpretations. The paper aims at evaluating a methodology for function point measurement based on the representation of the system through UML models: this methodology aims at providing a precise definition of the object of the measurement, as well as the measurement procedure and rules.

METHODS: An experimental application of the methodology is presented. A set of analysts (having different degrees of experience) were trained in the methodology and were then given the same requirements to model. The resulting models were measured by a few measurers, also trained in UML model-based counting.

RESULTS: The results show that the variability of the FP measure is small compared to the one obtained after applying “plain” FPA, as described in the literature. More precisely, whereas the influence of the modeller on the result appears to be negligible (i.e., a counter gets the same results from different models of the same application), the variability due to the measurer is more significant (i.e., different counters get different results from the same model), but still small when compared to the results reported in the literature on FPA.

CONCLUSIONS: The number of data points that we were able to collect was not big enough to allow reliable conclusions from a rigorous statistical viewpoint. Nevertheless, the results of the experiment tend to confirm that the considered technique decreases noticeably the variability of FP measures.

Keywords: Functional size measurement, Function Point Analysis, Measurement-oriented modelling, Measurement techniques, Experimental assessment.

1. INTRODUCTION

Function Point Analysis (FPA) is the most widely adopted method for measuring the functional size of programs. However, FPA suffers from a few relevant limitations. A first problem is that FPs are counted according to a set of informal rules that require human interpretation; moreover, the rules are defined in a rather fuzzy way, so that it is not always clear how every element of the requirements should be classified and counted. As a consequence, you need an expert, e.g., a person trained in the Function Point User Group (IFPUG) counting practice [1]. Counting rules apply to heterogeneous set of specification-related documents, written in any language or notation. In practice, FP counting has proven to be slow, expensive, and prone to a large variability.

Empirical data show that different counters can yield quite different measures of the same set of software requirements even in the same organization [6]; a 30% variance was observed within an organization while the observed difference was even greater across organizations [7]. Even according to data from the IFPUG the difference between counts provided by different certified experts for the same application may be up to 12% [10].

A second problem is that although the counting often requires a relevant effort to analyse several heterogeneous requirement documents in order to identify basic functional components (BFCs), such effort is not exploited to derive by-products, like a model of the measured software, or a satisfactory documentation of the measures. In fact, the identified BFCs are often not easy to trace back to elements of the requirements. Moreover, the effort devoted to understanding the requirements is not exploited to build any artefact that can be useful in the design and implementation phases.

A third problem is that the analyst defining the requirements and the measurer are two different persons. This makes it hard to insure that the correct functionalities are measured, and that the functionalities are measured correctly. In particular, there is no guarantee that the FP counter interprets the requirements correctly.

Several measurement techniques were proposed by researchers in order to solve some of the problems mentioned above. In this paper, we empirically evaluate one of such techniques, developed by two of the authors. The considered technique provides some methodological guidelines for building UML models in a measurement-oriented way. Since IFPUG FP counting is based on the identification of a set of elements that essentially correspond to system data and operations, the ability of UML to represent such information is exploited by establishing an explicit relationship between FP elements and UML language constructs. The method requires that either the analyst and the measurer build a UML model of the software to be measured together, or the analyst does it alone, according to well defined rules. This ensures that the model actually represents the functionality of the product and that the model includes all the elements needed for FPA. The actual measurement amounts to counting the elements of the model. The counting rules are quite straightforward and well defined, thereby contributing to decreasing the typical ambiguities of FPA.

Note that the objective of the considered technique is to improve the process of measuring FPs, i.e., of making FPA more efficient, effective, reliable, and repeatable. Aspects that might concern the definition [8] or the theoretic validity [9] of the Basic FP model, and related criticisms [11] are not addressed. In particular, although we are well aware of the problems with problem frames [6], the goal of this paper is not to change or improve the definition of FP, rather it is to evaluate whether the process of FP counting can be made less dependent on personal interpretation and choices, as well as on the type of documents and material describing the software requirements being measured.

The considered technique is expected to provide the following benefits:

- By making explicit the underlying model of the software application to be measured, it allows a precise definition of the BCFs, along with the way to identify, weight and count them.
- Guidelines help modellers in building UML models that are homogeneous with respect to the information contained and the level of detail, thus minimizing the variability of the FP measure.
- Since the goal is to improve the current industrial practice, IFPUG counting criteria are maintained, thus inheriting the ambiguities of the counting rules. However, with the proposed approach, most of the sources of ambiguities are limited to the modelling phase, when analysts and modellers can exploit their knowledge of the application domain in order to solve potential ambiguities.

This paper illustrates an experiment meant to evaluate the aforementioned technique, i.e., FP counting based on measurement-oriented UML models.

The paper is structured as follows: Section 2 briefly recalls the fundamentals of FP counting; Section 3 briefly describes measurement-oriented UML modelling, and FP counting based on the resulting UML models. Section 4 describes the experiment and analyses the results. Section 5 reports some criticisms to the function points method; these are illustrated to warn users of the evaluated technique about the limits of FPA. Section 6 draws some conclusions and sketches the future work.

2. FUNDAMENTALS OF FP COUNTING

Here we briefly recall the process of counting function points.

The Function Point method was originally introduced by Albrecht [2] to measure the size of a data-processing system from the end-user's point of view, in order to estimate the development effort. IFPUG FPA is now an ISO standard [4], as far as Unadjusted FPs – i.e. the actual functional size measure – are concerned. Throughout this paper we only consider unadjusted function points, even when the adjective 'unadjusted' is omitted.

The basic idea is that the 'amount of functionality' released to the user can be evaluated by taking into account the data elaborated by the application in order to provide the required functions, along with the transactions (i.e., operations that involve data crossing the boundaries of the application) through which the functionality is delivered to the user. Both data and transactions are evaluated at the conceptual level, in that they represent data and operations relevant to the user. Therefore, FPs are counted on the basis of the specifications of the user requirements.

The boundary indicates the border between the application being measured and the external applications and user domain. Representing the boundary is important, because identifying the operations and data that cross the boundary is fundamental in the computation of function points.

The core of the counting procedure consists in identifying and weighting data function types and transactional function types.

Data functions represent data that are relevant to the user and are required to perform some function. Data functions are classified into internal logical files (ILF), and external interface files (EIF). An ILF is a user identifiable group of logically related information maintained within the boundary of the application. The primary intent of an ILF is to hold data that are maintained by the application being counted and that are relevant with respect to the addressed problem. An EIF is similar to an ILF, but is maintained within the boundary of another application, i.e., it is outside the application being measured.

Transactional functions represent operations that are relevant to the user and cause input and/or output data to cross the boundary. Transactional functions represent elementary processes. An elementary process is the

smallest unit of activity that is meaningful to the user(s). The elementary process must be self-contained and leave the business of the application being counted in a consistent state. An elementary process is counted only if it is different from the others. Two elementary processes are different if they access a different set of ILFs and/or EIFs, or they access different data (possibly within the same ILF/EIF), or they are characterized by different processing logics. The processing logic indicates what is specifically requested by the user to complete an elementary process.

Transaction functions are classified into external inputs (EI), external outputs (EO), external inquiries (EQ). An EI is an elementary process whose primary intent is to maintain an ILF. EOs and EQs are elementary processes whose primary intent is to present information to a user. The processing logic of the elementary process counted as an EO contains relevant elaboration, while the processing logic of the elementary process counted as an EQ simply retrieves data from an ILF or EIF.

Every function (either data or transaction) contributes a number of FPs that depends on its “complexity”. The complexity of ILFs and EIFs is evaluated on the basis of Data Element Types (DETs) and Record Element Types (RETs). A DET is a unique, non-repeated field recognized by the user. A RET is a subgroup of the information units contained in a file. For transactions, the complexity is based on the number of DETs and File Type Referenced (FTRs). An FTR can be an ILF referenced or maintained by the transaction or an EIF read by the transaction. The DETs considered are those that cross the application boundary when the transaction is performed.

The weighting of function types on the basis of their complexity is done according to tables (not reported here). Finally, the FP number is obtained by summing the weighted function types, according to Table 1. For instance, the size of a system featuring 2 low complexity ILF, a high complexity EIF, 3 low complexity EI and a medium complexity EO is $2 \times 7 + 1 \times 10 + 3 \times 3 + 1 \times 5 = 38$ FP.

TABLE 1: Function type weight according to complexity

Function type	Low	Medium	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

As already mentioned, the definition of the function points is not always precise, thus leaving space for subjective interpretations. Consider for instance a program to play tic-tac-toe: entering a user’s move causes the computation of the computer’s move, the storage of both moves in the game ILF and outputting the computer’s move. According to IFPUG rules, this transaction can be classified both as EI or EO. In fact, in order to decide the transaction type, we should identify its main purpose, which is very hard in this case: what is more important, updating the internal game state (without which the program would be unable to work) or outputting the moves (without which the program would be useless)?

3. MEASUREMENT-ORIENTED UML MODELLING

The proposed process has two phases:

1. A UML model of the system to be measured is built, according to the guidelines described in Section 3.1.
2. The UML model is analysed and FPs are counted, according to the rules illustrated in Section 3.2.

3.1 Measurement-oriented modelling

The measurement-oriented modelling guidelines are published elsewhere [12]; therefore we report here only a very brief description of the most relevant parts of the methodology.

The first objective of the model is to represent the application boundaries and the external elements that interact with the system to be measured.

UML provides Use Case Diagrams, which are well suited for our purposes. In fact:

- Use Case Diagrams clearly indicate the boundaries of the application.
- Use Case Diagrams represent –as actors– the elements outside the boundary with which the application interacts.
- Most important, Use Case Diagrams show the transactions. Representing each elementary process as a use case is both easy and consistent with the typical usage of use cases.

Figure 1 reports a use case diagram. The application boundary is explicitly represented, the functionalities required from the system are indicated as use cases, and the external elements interacting with the system are shown. Among these, we may have the operator, I/O devices, and external sources of data (typically maintained/supplied by external applications).

FIGURE 1: Use case diagrams represent boundaries and user-perceivable functionality

Component diagrams are used to represent EIFs and ILFs, while the structure of transactions –needed for comparing transactions and to weight them– is represented via sequence diagrams (SDs). Component diagrams provide suitable features:

- ELF and ILF can be represented as components, as well as the other external elements that interact with the application.
- Components can be stereotyped, in order to clarify their nature. We defined the following stereotypes: <<Logic data>>, representing ILFs (inside the boundary) and EIFs (outside the boundary), and <<I/O>> (possibly specialised into <<I/O::IN>>, <<I/O::OUT>>, <<I/O::InOut>>) to represent external elements that provide input to or consume output generated from the application.
- Interfaces indicate the operations provided and required by every component. They identify transactions (i.e., operations provided by the component representing the application) and the operations that can be performed on ILFs and EIFs.

Note that components are intended as conceptual units. This is coherent with the fact the model represents the requirements for the application, where physical software components are generally not to appear.

FIGURE 2: Component diagram representing the system to be measured

Figure 2 reports the component diagram for the System to be measured. It is possible to see that there is one component that represents the application; therefore the application boundaries are explicit in this diagram, too. The actors that appeared in the use case diagram are still present as components: two are classified as <<I/O>> elements, while the other two are classified as <<Logic data>> and, being outside the boundaries, are EIFs. The operations in the interfaces of the application component represent transactions; the parameters of these operations (not reported in Figure 2) specify the data that cross the boundaries. There is an operation for each use case appearing in Figure 1.

ILFs are represented as <<Logic data>> sub-components of the application component. Although IFPUG defines ILFs in a rather imprecise manner, we expect analysts to be able to identify and classify the 'user identifiable group of information' without big troubles.

The content of each EIF and ILF is described in a separate component diagram (see for instance Figure 3). This information is necessary to weight correctly the data functions.

FIGURE 3: Component diagram representing an ILF

In general, a data function corresponds to a class. A relevant exception is given by clusters of classes that are connected by composition or generalisation relations, as in Figure 3. In some cases, even associations are strong enough to let us consider the associated classes as one ‘user identifiable group of information’.

For the sake of FPA, it is necessary that for each data function, the following information is reported:

- Classes that represent the user relevant data groups, and subgroups, if present.
 - Attributes that represent the elementary data elements which are meaningful to the user.
 - Relationships between classes, especially as far as composition and generalization are concerned.
- The rest of the information required for FPA is given by sequence diagrams, which allow us to characterize elementary processes that are potential transactions as follows:

- The correspondence of elementary processes to user-relevant functions – as described in the use case diagram – must be explicit.
- Logical files that are read or written have to be identified. Operations on files must be recognizable as read only, mainly write, or read/write (this is made possible by stereotyping the operations that appear in the interfaces in the component diagrams).
- Data that cross the boundary have to be explicitly represented (typically as operations’ arguments).
- Processing must be described at a level of detail sufficient to verify whether two elementary processes are equivalent.

Note that these sequence diagrams lay completely in the realm of requirements, since they represent only the data exchanged with the outside and the access to internal logical data (logical data are meaningful to the user, and explicitly described in the user requirements). As to the description of the processing contained in the SD, it provides just the minimum information required by FPA; no design element needs to be considered.

FIGURE 4: Sequence diagram describing a transaction

The sequence diagram in Figure 4 illustrates the elementary process through which a transaction is performed. It is easy to see that the diagram achieves the objectives we set for sequence diagrams:

- The elementary process represented corresponds to the FunctionA that appears in the use case diagram and in the System’s interface.

- Access to ILT SystemData is clearly indicated. Similarly, the involvement of the External data supply is explicitly represented.
- The data that cross the boundary are described: they are the arguments of FunctionA operation, the data read from the External data supply, the arguments of the Write operation, and the acknowledgement message.

3.2 Counting Function Points

The FP counting procedure is defined with respect to the UML model with the following goals: it must be coherent with the principles reported in the IFPUG counting manual [1]; it must be precise, without ambiguities, thus leaving no space to the counter for interpretation; it must require little effort; it must be executable by people without big skill in FP counting and with little knowledge of the requirements. The prospect is to make the procedure executable automatically, thus definitively removing the subjectivity of FP counting.

3.2.1 Counting EIFs and ILFs

Identifying ILFs and EIFs is immediate: both are components stereotyped <<Logic data>>; ILFs are within the boundaries of the application, i.e., in the application component, EIFs are outside. In order to weight ILFs and EIFs, we need to measure their RETs and DETs.

As far as RETs are concerned, the IFPUG directive is “Count a RET for each optional or mandatory subgroup of the ILF or EIF.” Since in our models the data subgroups are represented by classes, in general we count a RET for every class in the data component. Abstract classes are not counted. Classes belonging to the same composition are counted as one RET. Details of RET counting are in [12].

Counting the DETs is relatively simple: we count a DET for each non-repeated attribute of the class(es) belonging to the data component.

Therefore, SystemData in Figure 3 has two RETs (corresponding to Class_Ax and Class_Ay) and 8 DETs. It is thus a low complexity ILF, and weights 7 FPs.

3.2.2 Counting transaction functions

Transactions correspond to the elementary processes described by the sequence diagrams.

The first step consists in classifying each transaction as an EI, EO or EQ. The criteria for distinguishing the different types of transactions, as reported in [1], are not always effective. Although in general stereotyping the operations as read only, mainly write or read/write helps in deciding the nature of the transaction, there are cases when it is not easy to distinguish EIs from EOs, or EOs from EQs. Subtle cases can occur, when deciding what is the main intent of the function or recognizing the existence of internal elaboration is quite hard. A deep understanding of the system is required in order to take the proper decision. Therefore, the modeller has to put in the model as much FPA-relevant information as possible: he/she is required to annotate the diagram. In Figure 4 the Write operation is labelled as the main intent of the transaction: this means that it is either an EO or an EQ. The annotation of the Compute operations tells us that the transaction is an EO.

The second step consists in counting the FTRs and the DETs.

Counting the FTRs is immediate: we just have to count how many ILFs and EIFs are referenced. In the SD in Figure 4, the External data supply and SystemData are referenced, thus FTR=2.

The DETs to be considered are the ones that cross the boundary: the arguments of FunctionA operation, the data read from the External data supply, the arguments of the Write operation, and the acknowledgement message. Supposing that $5 < \text{DET} < 20$, the function is an average complexity EO, and weights 5 FP.

4. THE EXPERIMENT

4.1 Goals

The idea underlying measurement-oriented UML models is that an FP counter must be given a precise description of the requirements of the application, and that such description has to contain all the elements needed for performing FPA according to the IFPUG criteria.

By unifying the notation employed, and the modelling concepts, as well as by getting the modeller directly involved into the description of the system and in the identification and description of BFCs, the methodology aims at decreasing the uncertainty about what has to be measured and what are the associated features.

As a consequence, a critical question to be answered is the following: *can we expect that different analysts provide the same description (or equivalent ones) of given system requirements?* If this condition did not hold, we would have just moved the complexity and variability of FPA from the counting phase to the modelling phase.

In order to test this assumption, we invited several modellers to independently specify the same application, following the measurement-oriented modelling methodology described in Section 2, and measured the models they provided.

Actually, also the counting phase is subject to interpretations – being based on the IFPUG criteria – and errors. It is therefore interesting to consider whether different FP counters would provide different measures of the same model. Accordingly, we had every model measured by a few different counters.

4.2 Organization of the experiment

4.2.1 The sample application

In order to test the methodology, we chose an application having the following characteristics:

- It is a rather simple information system. Thus, it is a type of application that is traditionally considered suitable for FP counting. Moreover, the application domain is well known by the participants, thus understanding the requirements does not pose any additional difficulty.
- It is a small application. It was necessary to limit the size of the application in order to keep the effort required to the participants as low as possible. In fact, an application requiring too a high modelling effort would have surely discouraged several participants, especially the professionals.

4.2.2 The experiment

The experiment was organised as follows.

The measurement-oriented modelling technique was illustrated to a set of persons reasonably familiar with UML modelling. These people had different experience levels. In fact, the modelling experiment was carried out in two phases: In the first phase only relatively senior persons were involved; in the second one, we only involved undergraduate students. In particular,

- Modellers 1 and 2 are analysts with more than 20 year long experience;
- Modellers 3 and 5 are PhD students;
- Modeller 4 is a young professional (about 5 year experience);
- Modeller 7 is a post-doc.
- Modellers 6 and 8 are undergraduate students.

In both the modelling phase and the measurement phase, the level of knowledge, training and experience of the participants could affect the results. We addressed this issue by involving people with different characteristics, in order to test whether the sensitivity of the proposed technique to the personal differences is actually little. In fact, one of the main goals of the measurement-oriented modelling technique was just to get similar models (i.e., containing the same information, at the same level of abstraction) from different people, and, similarly, to get similar counts from different counters. If the results of the experiment did not confirm this hypothesis, we should have analysed the dependence of the results on people's culture and training. For this purpose we could rely on the fact that we personally knew every participant (it was a small set, unfortunately). Those interested in replicating this experiment could consider characterizing modellers and counters according to knowledge, training, experience, etc.

The participants operated in isolation, without any exchange of information concerning the construction of the model.

The measurement-oriented modelling technique was illustrated by means of a one-hour presentation. For reference, a document describing the methodology along with two examples of application was given to the participants.

The informal requirements of the application to be modelled were given to all participants through a written document. The requirements were expressed by a mix of natural language and traditional diagrams (namely, E/R and data flow diagrams) that were expected to be fairly well known by most participants. The participants could ask for details and clarifications if they had any doubt, but nobody did actually take advantage of this possibility. We can therefore state that the provided informal requirements were clear enough to support the modelling activity. The requirements were specifically written in a way so as not to suggest how to build the model.

The participant produced a UML model representing the requirements of the system, according to the measurement-oriented modelling technique. Unfortunately, only a subset of the people invited to participate in the experiment did actually deliver a model. In particular, we received a first set of five models from the most experienced people, who were also able to dedicate to the modelling task enough time and attention. In order to increase the number of available models, we then invited another set of people – mainly students – to participate in the experiment. From this second round we obtained three more models.

During the first phase we received a couple of models containing errors (i.e., not matching the user requirements). In such cases we asked the modellers to correct the errors. This was not possible in the second phase, resulting in lower quality models, as we shall see in a moment.

UML models were collected, and their functional sizes were evaluated following the procedure sketched in Section 3.2. In order to make the evaluations reasonably independent of the counter, all the models were evaluated by two or three counters. The measurers had various levels of experience in FP counting: Measurer A has a couple of years' experience, Measurer B a few months, Measurer C just a couple of weeks. The counting rules are quite straightforward, thus we expected that the different measurers would yield similar results. Even though two or three

is too small a number of measurers to support any reliable consideration, we also got some useful indications on this issue.

Measurer B was not available for the second phase; therefore, Models 6 to 8 were measured only by Measurers A and C.

In order to get a reference size measure, an experienced FP counter measured the requirements of the system according to the traditional IFPUG procedure (i.e., without taking into consideration any UML model). The result was that the size of the sample application is 67 FP.

4.3 Results

The results of our experiment are collected in Table 2.

TABLE 2: Results of the experiment

Model	Measurer A			Measurer B			Measurer C		
	Data	Trans	FP	Data	Trans	FP	Data	Trans	FP
1	33	33	66	33	28	61	33	34	67
2	33	33	66	33	31	64	33	33	66
3	33	31	64	33	28	61	33	31	64
4	33	33	66	33	28	61	33	33	66
5	33	33	66	33	30	63	33	34	67
6	26	23	49	N/A	N/A	N/A	26	23	49
7	40	28	68	N/A	N/A	N/A	40	30	70
8	33	31	64	N/A	N/A	N/A	33	33	66

Since we do not have enough data to draw reliable statistical conclusions, we make just a few qualitative considerations about the results.

A first observation is that the number of counted function points is similar for all models, with two noticeable exceptions:

- The model of Modeller 6 yielded a sensibly smaller number of function points. We analysed such model in order to understand why, and we discovered that Modeller 6 provided a simplified description of the requirements, in the sense that the described functionalities are different from the required ones, and this also implied the usage of less data. The simplification of the user requirements resulted – quite obviously – in a smaller number of FPs. On one hand, this result is coherent with the method: a set of simpler requirements has a smaller size in FPs. On the other hand, a lesson learned is that in order to apply the method, reasonably experienced analysers are needed, which are able to represent precisely and completely the user requirements.
- The model of Modeller 7 yielded a number of data function points (i.e., FP counted on ILF and EIF) larger than all the other models. On the contrary, model 7 yielded a size of transactions (i.e., FP counted on EI, EO and EQ) smaller than the other models (except model 6). These two errors compensate each other, so that the overall size (68 or 70 FP, depending on measurer) is very close to the expected size (67 FP).

In conclusion, we should consider model 6 as a definite outlier, while model 7 is not distinguishable from the other models if we just look at the overall size. Model 7 should in fact be viewed as an outlier, if we consider data and transactional size separately.

Excluding model 6, we have that for Measurer A the average size of the models is 65.7 FP while, for Measurer C it is 66.6 FP. For measurer B the average (computed only on the first 5 models) is 62 FP. Considering that the size evaluated by a certified FP counter would most probably be 67 FP, we have a quite good result: both the differences among the counters, and their “error” with respect to the actual presumed size is reasonably small.

By looking at the distribution of the values in the columns, i.e., by considering how each modeller evaluated his/her set of models, we can see that the variance is also reasonably small: the size ranges from 64 to 68 FP for Modeller A, from 61 to 64 FP for Modeller B, from 64 to 70 FP for Modeller C. This observation allows us to conclude that in our experiment the modellers had a minor influence in determining the variability of final FP counting.

In order to see whether significant FP variability can be caused by different counters, we can also examine the FP values contained in Table 2 in a row-wise fashion. We have that in general different measurers size the same model differently; however the maximum difference between two sizings of the same model (6 FP, the difference between the evaluations of model 1 performed by Measurers B and C) is reasonably small. This suggests that there is still a mild dependence of FP size on the measurer.

Another interesting observation is that all measurers counted exactly the same number of data function points for every model, i.e., all the counters agreed on the number of data FP *for every model*. On the contrary, there are several models whose functional size was sized differently by different counters. For instance counter A, B and C evaluated that the transactions in model 1 are 33, 28 and 34 FP, respectively. This result probably indicates that

the considered methodology provides indications for counting data functions that are enough precise, well understood, and easy to apply, thus causing very little variability in the measure of the data size. On the other hand, the indications for counting transactional functions are apparently not as good as to let different counters produce the same result. Thus in future work we plan, among other things, to improve the directives for counting transactional function points, with the goal of decreasing the variability of this measure.

4.4 Comparison with the results reported in the literature

Since function point counting involves judgment on the part of the counter, the resulting measures are affected by some variability. Chris Kemerer reported a 12% difference for the same product by people in the same organization [13]. Graham Low and Ross Jeffery reported a 30% variance within an organization, which rose to even more than 30% across organizations [7].

Even according to data from the IFPUG the difference between counts provided by different certified experts for the same application may be up to 12% [10].

According to the results of our experiment, the measurement of function points based on properly constructed UML models seems to yield better performances. It seems –although the little number of data points does not allow reliable conclusions– that different measurers tend to yield similar measures of a given model: in the experiment the *maximum* difference between the evaluations of the same model performed by different counters was 6 FP (for a 67 FP application), while in most cases it was less than 3 FP.

Also the variability due to the modellers, i.e., to the way user requirements are described, appears to be fairly small.

5. FUNCTION POINTS ASSESSMENTS AND CRITICISMS

The evaluated methodology [12] aims at making the application of FPA easier and more reliable. The goal is to ease the application of FPA in industry, since FPA is a de-facto industrial standard (as well as a de-jure one [4]).

However, FPA users must be warned about the limits of the FPA. In fact, Function Points Analysis has been widely criticized in the literature. In [6] Kitchenham highlights most of the criticisms of Function Points methodology; among these, the most relevant ones (besides those we already mentioned) are:

- Measures are used in an inconsistent way. Ordinal scale [17] measures are added up together.
- Predictive models based on Function Points are not stable for different datasets. That is, a correlation between effort and Function Points can be found, but the correlation change widely when changing the datasets.
- There is no evidence that the technology adjustment factors improve the accuracy of predictive models based on Function Points. This observation has been confirmed by many researches, see for instance [14].
- Function counts are not a technology independent measure [19][20].

Note that neither the definition of the model-based FP measurement described in [12] nor the work reported here address the problems mentioned above.

In order to overcome the aforementioned limits and criticisms, several variations of the Function Points methodology have been proposed in literature. In [14] the Function Points methodology is compared to SPQW/20, a variation proposed by Jones [18]; the findings are not promising: in the controlled experiment that has been conducted it seems that both approaches suffer from the very same problems. In [21] Antoniol, Fiutem and Lokan propose a variation of the methodology based on Object Oriented approaches; however, the resulting measures are not compatible with function points a la IFPUG. In [22] Wittig and Finnie propose to use Neural Networks to correlate Function Point measures and actual effort; the approach seems promising, even if Neural Networks requires big datasets for calibration before actual use.

Also COSMIC FPs [25] were defined in order to overcome some limitations of the Function Points; they advocate the derivation of a model from user requirements, before performing the actual measurement. However, some criteria for building the model (e.g., the identification of “data groups”) are not unambiguous, and the modelling notation is neither standard nor sufficient to support requirements analysis.

Function Points approach suffers from many problems, but they are widely considered among the best estimation approaches currently available; [23] and [24] evaluate and compare various estimation approaches, although in a very informal manner.

6. CONCLUSIONS AND FUTURE WORK

This paper reports an experiment whose goal is to assess the effectiveness of a technique [12] meant to improve the practice of function point counting. This technique proposes to build UML models that contain the information needed for FP counting. Then, the measurement is performed in a relatively straightforward way by counting the relevant elements of the UML diagrams. The evaluated technique aims at making it easier to both describe what has to be measured (i.e., the user requirements) and carry out the actual measurement. Besides, since the model of the application to be measured is built by (or with the help of) the analyst of the application, it is expected that the object of the measurement is precisely defined and does not leave to much room for interpretation. As a consequence, this reduces the variability of the size measurement. Similarly, the simplicity of the counting rules is also expected to contribute to decrease the variability of the measurement.

In order to validate these expectations, an experiment was carried out: the same application was modelled by several analysts and the resulting models were measured by a few measurers. Unfortunately the number of data

points that we were able to collect was not big enough to allow reliable conclusions from a rigorous statistical viewpoint. Nevertheless, the results of the experiment tend to confirm that the considered technique noticeably decreases the variability of FP measures.

Even though the sample size was too small for proper statistical analysis, we believe there are at least two good reasons for spreading the results of this experiment: one is that a validation –though partial and qualitative– of the model-based measurement technique was needed, and the experiment provided it; the other is that the reasonably good results and the clear need to perform further investigations could induce other researchers to replicate the experiment (the descriptions of the methodology and the requirements of the application used in the experiment are available from the authors).

Finally, the validation of the model-based measurement technique [12] requires also that it is tested on larger applications (e.g., having size > 200 FP). This activity will be topic of future work.

ACKNOWLEDGMENTS

Many thanks to the colleagues, friends and students that participated in the experiment, thus making it possible to evaluate the FP counting technique based on measurement-oriented UML models.

REFERENCES

- [1] International Function Point Users Group, "Function Point Counting Practices Manual - Release 4.2", 2004.
- [2] A.J. Albrecht, "Measuring Application Development Productivity", Joint SHARE/GUIDE/IBM Application Development Symp., 1979.
- [3] V. Del Bianco e L. Lavazza, "An Empirical Assessment of Function Point-Like Object-Oriented Metrics", 11th Int. Software Metrics Symposium, Como, September 2005.
- [4] ISO/IEC 20926: 2003, "Software engineering -- IFPUG 4.1 Unadjusted functional size measurement method -- Counting Practices Manual", ISO, 2003.
- [5] D. Garmus, D.P. Herron "Function Point Analysis: Measurement Practices for Successful Software Projects", Addison-Wesley Information Technology Series, 2001.
- [6] B. A. Kitchenham, "The problem with function points", IEEE Software, 14(2), 1997.
- [7] G. C. Low and D. R. Jeffery, "Function points in the estimation and evaluation of the software process", IEEE Transactions on Software Engineering, 16(1), 1990.
- [8] Cantone G., "Measure-driven Processes and Architecture for the Empirical Evaluation of Software Technology", Journal of Software Maintenance, Vol. 12, No. 1, 2000.
- [9] Briand, L., Morasca, S., Basili, V. R., "Property-Based Software Engineering Measurement", IEEE TSE, Vol. 22, No. 21, 1995.
- [10] L. Buglione, "Misurare il software", Franco Angeli, Milano, Italy, 2003 (in Italian).
- [11] S. Oligny, A. Abran, "On the compatibility between Full Function Points and IFPUG Function Points", 10th ESCOM, Herstmonceux Castle, England, April 1999.
- [12] L. Lavazza, V. del Bianco, C. Garavaglia, "Model-based Functional Size Measurement", To be presented at ESEM 2008, 2nd International Symposium on Empirical Software Engineering and Measurement, Incorporating ISESE and Metrics, Kaiserslautern, Germany. October 9-10, 2008.
- [13] C.F. Kemerer, "An Empirical Validation of Software Cost Estimation Models," Comm. ACM, Vol. 30, No. 5, 1987, pp 416-429.
- [14] D.R. Jeffery, G.C. Low, and M. Barnes, "A Comparison of Function Point Counting Techniques," Software Engineering, IEEE Transactions on, vol. 19, 1993, pp. 529-532.
- [15] M. Shepperd and C. Schofield, "Estimating Software Project Effort Using Analogies," IEEE Trans. Softw. Eng., vol. 23, 1997, pp. 736-743.
- [16] C.F. Kemerer, "Reliability of function points measurement: a field experiment," Commun. ACM, vol. 36, 1993, pp. 85-97.
- [17] N.E. Fenton and S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, Revised, Course Technology, 1998.
- [18] T. C. Jones, "A short history of function points and feature points," Software Productivity Research Inc., 1988.
- [19] C. R. Symons, "Function point analysis: Difficulties and improvements," IEEE Trans. Software Eng., vol. 14, no. 1, pp. 2-11, 1988.
- [20] J. Vemer, G. Tate, B. Jackson, and R. Hayward, "Technology dependence in function point analysis: A case study and critical review," Dep. Comput. Sci., Massey Univ. Sept., 1988.
- [21] G. Antoniol, R. Fiutem, and C. Lokan, "Object-Oriented Function Points: An Empirical Validation," Empirical Softw. Engg., vol. 8, 2003, pp. 225-254.
- [22] G. Wittig and G. Finnic, "Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort," Australasian Journal of Information Systems, vol. 1, 1994.
- [23] S. MacDonell, "Comparative review of functional complexity assessment methods for effort estimation," Software Engineering Journal, vol. 9, 1994, pp. 107-116.
- [24] P.D. Chatzoglou and L.A. Macaulay, "A review of existing models for project planning and estimation and the need for a new approach," International Journal of Project Management, vol. 14, Jun. 1996, pp. 173-183.
- [25] Abran, A. et al., "The COSMIC Functional Size Measurement Method Version 3.0 Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761: 2003)", September 2007