On the Ability of Functional Size Measurement Methods to Size Complex Software Applications

Luigi Lavazza   Sandro Morasca   Davide Tosi
Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese, Italy
{luigi.lavazza, sandro.morasca, davide.tosi}@uninsubria.it

Abstract—**The most popular Functional Size Measurement methods, namely IFPUG Function Point Analysis and the COSMIC method, adopt a concept of "functionality" that is based mainly on the data involved in functions and data movements. Neither of the mentioned methods takes directly into consideration the amount of data processing involved in a process. Functional size measures are often used as a basis for estimating the effort required for software development, and it is known that development effort does depend on the amount of data processing code to be written. Thus, it is interesting to investigate to what extent the most popular functional size measures represent the functional processing features of requirements and, consequently, the amount of data processing code to be written. To this end, we consider a few applications that provide similar functionality, but require different amounts of data processing. These applications are then measured via both functional size measurement methods and traditional size measures (such as Lines of Code). A comparison of the obtained measures shows that differences among the applications are best represented by differences in Lines of Code. It is likely that the actual size of an application that requires substantial amounts of data processing is not fully represented by functional size measures. In summary, the paper shows that not taking into account data processing dramatically limits the expressiveness of the size measures. Practitioners that use size measures for effort estimation should complement functional size measures with measures that quantify data processing, to get precise effort estimates.**

Keywords- functional size measurement; Function Point Analysis; IFPUG Function Points;COSMIC method.

I. INTRODUCTION

The most popular Functional Size Measurement (FSM) methods, i.e., IFPUG (International Function point User Group) [1][2][3] and COSMIC (Common Software Measurement International Consortium) [4]– adopt a concept of "functionality" that is based mainly on two elements:

– the processes, named Elementary Processes (EP) in IFPUG and Functional Processes (FPr) in COSMIC;
– the data that cross the boundary of the application being measured or are used (i.e., read or written) in the context of a process.

Quite noticeably, neither method satisfactorily considers the amount of data processing involved in a process. As a matter of fact, Function Point Analysis proposes an adjustment of the size based on the complexity of data processing, but, as discussed in Section VII, quite imprecisely and ineffectively, while the COSMIC method does not take the amount of data processing into account at all.

The goal of the paper is to provide evidence, by using an example, that not considering data processing dramatically limits the expressiveness of functional size measures.

The core of the paper can be described as follows:

– Two applications are specified. These applications are similar with respect to the aims and functionality offered to the user, but they are very different in the amount and complexity of the processing required.
– The two applications are modeled and measured according to the IFPUG and COSMIC rules.
– It is highlighted that the two applications have the same functional size measures, even though the amount of functionality to be coded in the two cases is enormously different.
– In fact, when measured via Lines of Code, it is apparent that the implementations of the two applications have quite different sizes. The reason is that more data processing clearly requires more code.

The conclusion is that using only the functional size to estimate development effort is likely to yield huge errors for complex applications. Since size measures are used for effort estimation, using functional size measures to size complex applications (i.e., programs that require a substantial amount of data processing) may lead to large (and dangerous) effort underestimations.

The paper is structured as follows. Section II reports a few basic concepts of functional size measurement. Section III illustrates the case studies used in the paper. Section IV describes the models and measures of the considered applications: the collected measures are then compared in Section V. Section VI discusses the alternatives that should be considered for complementing standards functional size measures with measures that represent data processing. Section VII accounts for related work. Finally, Section VIII draws conclusions and briefly sketches future work.

II. FSM CONCEPTS

Functional size measurement methods aim at providing a measure of the size of the functional specifications of a given software application.

Here, we do not need to explain in detail the principles upon which FSM methods are based. Instead, it is important for our purposes to consider what is actually measured, i.e., the model of software functional specifications that is used by the Function Point Analysis (FPA) and COSMIC methods.

The model used by FPA is given in Figure 1. Briefly, Logical files are the data processed by the application, and transactions are the operations available to users. The size measure in Function Points is computed as a weighted sum of the number of Logical files and Transactions. The weight of logical data files is computed based on the Record Elements Types (RET: subgroups of data belonging to a data file) and Data Element Types (DET: the elementary pieces of data). The weight of transactions is computed based on the Logical files involved –see the FTR (File Type Referenced) association in Figure 1– and the Data Element Types used for I/O.

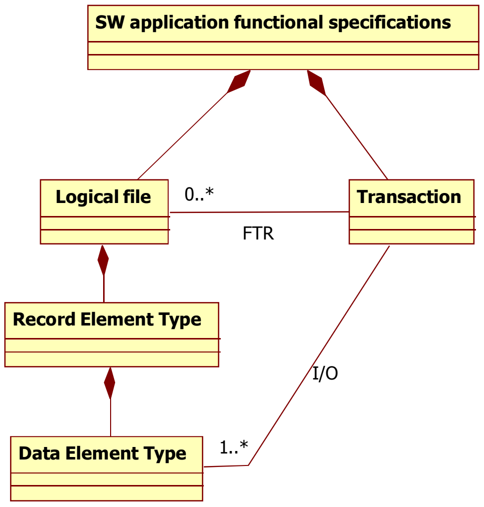

Figure 1. The model of software used in Function Point Analysis.

It is possible to see that in the FPA model of software, data processing is not represented at all.

The model used by COSMIC is given in Figure 2. The size of the functional specification expressed in COSMIC function points (CFP) is the sum of the sizes of functional processes; the size of each functional process is the number of distinct data movements it involves. A data movement concerns exactly one data group.

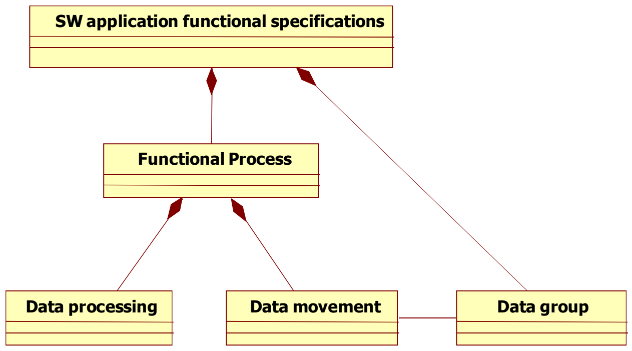

Figure 2. The model of software used by the COSMIC method.

Neither data groups nor data processing are directly used in the determination of an application's functional size. In particular, data processing is not measured at all. The COSMIC method assumes that a fixed amount of data processing is associated with every data movement; however, it is not so, in the examples considered in this paper.

III. CASE STUDIES

In this section, we describe the functional specifications of the two software applications that will be used to test the functional sizing ability of FPA and COSMIC.

The chosen applications are programs to play board games against the computer. They are similar with respect to the provided functionality, but require different amounts of data processing.

The specifications that apply to both applications are as follows:

– The program lets a human player play against the computer.
– The program features a graphical interface in which the game board is represented.
– The player makes his/her moves by clicking on the board. Illegal moves are detected and have no effect. As soon as the human player has made a move, the computer determines its move and shows it on the board.
– When the game ends, the result is shown, and the player is asked if he/she wants to play another game.

A. *A Software Application to Play Tic-tac-toe*

Tic-tac-toe is a very simple, universally known game. It is played on a 3×3 board, as shown in Figure 3. Each player in turn puts his/her symbol in a free cell. The first player to put three symbols in a row (horizontally, vertically or diagonally) wins. When the board is filled and no three-symbol row exists, the match is tie.

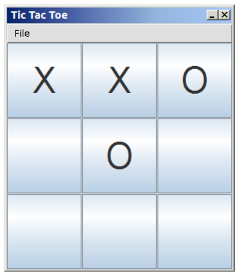

Figure 3. Tic Tac Toe playing board.

Playing Tic-tac-toe is very simple. In fact, to play optimally, a software program has just to evaluate the applicability of the following sequence of rules: the first applicable rule determines the move:

1) If there is a row such that two cells contain your symbol, and the third cell X is empty, put your symbol in the free cell X.
2) If you are the first to move and this is your first move, put your symbol in the central cell.
3) If there is a row in which your opponent has two symbols and the third cell X is free, put your symbol in the free cell X.
4) If there is a free cell X such that putting your symbol there results in two rows, each one having two cells occupied by your symbol and the third cell free, put your symbol in cell X.
5) If there is a row in which you have one symbol and the other two cells X and Y are free, put your symbol in cell X or in cell Y.

The code that implements the playing logic described above is very simple and very small: we can expect that a few tens of lines of code are sufficient to code the game logic.

B. A Software Application to Play "five in a row"

Five in a row (aka Gomoku) can be seen as a generalization of Tic-tac-toe. In fact, it is played on a larger board (typically 19×19, as in Figure 4) and the aim of the game is to put five symbols of a player in a row (horizontally, vertically or diagonally).

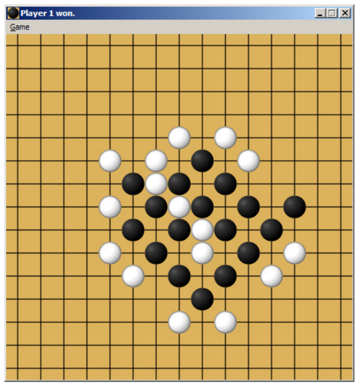

Figure 4. Gomoku playing board.

The functional specifications of Gomoku are exactly the same as the specifications of Tic-tac-toe, except that

a) The size of the board is larger
b) The number of symbols to put in a row is 5 instead of 3.

The combinations of symbols and free cells that can occur in a Gomoku game are many more than in a Tic-tac-toe game. Accordingly, a winning strategy is much more complex, as it involves considering a bigger graph of possibilities.

As a matter of fact, Gomoku has been a widely researched artificial intelligence research domain, and there are Gomoku professional players and tournaments.

Accordingly, we can safely state that Gomoku is a much more complex game than Tic-tac-toe, and it requires a huge amount of processing, so that the machine can play at a level that is comparable with that of a human player.

On the contrary, Tic-tac-toe is a very simple game: you do not need to be particularly smart to master it and always play perfectly.

IV. Application Sizing

A. A Software Application to Play Tic-tac-toe

Let us measure the Tic-tac-toe specifications given in Section III.A above, starting with IFPUG Function Points.

The software model to be used involves just a Logical data file: the board and a matrix of cells, each having one of three possible values (circle, cross, free).

The software model to be used involves the following elementary processes:

– Start a new game.
– Make a move.

It is not necessary to consider details (RET, DET) to see that the Logical data files is a simple Internal Logical File (ILF), contributing 7 FP.

Similarly, it is not necessary to consider details (FTR, DET) to see that:

– Start a new game is a simple External Input (EI), contributing 3 FP.
– Make a move is a simple external output, contributing 4 FP. One could wonder if this operation should be considered an input (because the move involve inputting a position) or an output (because of the computation and visualization of the move by the computer). We consider that the latter is the main purpose of this transaction, which is thus an external output.

In summary, the FPA size of the Tic-tac-toe application is 14 FP.

The COSMIC functional processes of the application are the same as the FPA elementary processes. When measuring the application using the COSMIC method, we have to consider the data movements associated with each functional process:

– Start a new game involves clearing the board and possibly updating it, if the computer is the first to move (a Write) and showing it (a Read and an Exit). Therefore, this functional process contributes 3 CFP.
– Make a move involves entering a move (an Entry), updating the board with the human player move (a

Write), reading it (a Read), and then updating it again with the computer move and showing it (an Exit). In addition, if a move concludes the game, the result is shown (an Exit). Therefore, this functional process contributes 5 CFP.

In summary, the COSMIC size of the Tic-tac-toe application is 8 CFP.

Since we are also interested in indications concerning the amount of computation performed by the application, we selected an open source implementation of Tic-tac-toe and measured it.

To evaluate the "physical" size of the Tic-tac-toe application, we looked for an open source application that implements the specifications described above. One such application is the program available from [8].

The main measures that characterize the code are given in TABLE I.

TABLE I. MEASURES OF THE TIC-TAC-TOE APPLICATION CODE

Measures	Tic-tac-toe [8]	
	Total	***AI part***
LoC	172 (118 statements)	66 (52 statements)
McCabe	3.6	5
Num. classes	2	1
Num. methods	17	7

In TABLE I (and in TABLE II), column "AI part" indicates the measures concerning exclusively the part of the code that contains the determination of the computer move.

In the LoC line, we reported both the number of lines and the number of actual statements. The latter is a more precise indication of the amount of source code. We also reported the mean value of McCabe complexity of methods.

B. *A Software Application to Play "five in a row"*

The functional size measures of the Gomoku application are exactly the same as the measures of the Tic-tac-toe application. In fact, the specifications of the two applications are equal, except for the board size and winning row size, which do not affect the measurement, because both IFPUG FPA and COSMIC consider data types, not the value or number of instances.

As for Tic-tac-toe, we selected an open source implementation of Gomoku and measured it. More precisely, to take into account that a programmer may aim at developing a program capable of more or less sophisticated "reasoning," we considered a few different implementations of Gomoku.

In this case, to evaluate the "physical" size of the application, we also looked for an open source application implementing the specifications described above. One such application is the Gomoku application available from [9].

The main measures that characterize the code are given in TABLE II.

TABLE II. MEASURES OF THE GOMOKU APPLICATION CODE

Measures	Gomoku [9]	
	Total	***AI part***
LoC	832 (395 statements)	425 (234 statements)
McCabe	3	5.95
Num. classes	12	3
Num. methods	63	21

Measures in TABLE II were derived using the same tools and have the same meaning as the measures in TABLE I.

V. COMPARISON OF MEASURES

The measures reported in the previous section show that we can have two applications that have the same functional size, but very different code size (the Gomoku applications are over four times as big as the Tic-tac-toe application). Considering the nature of these applications, the difference in code is largely explained by the different amount of processing required. In the case of Tic-tac-toe, the number of possible moves is very small, as is the number of different possible configurations that can be achieved by means of a move: hence, every move computation has to explore a very small space. The contrary is true for the Gomoku application. The consequence is that Gomoku requires an amount of code devoted to move computation that is over 6 times the code required by Tic-tac-toe (or 4.5 times, if we consider the number of statements instead of LoC).

These observations suggest two important considerations:

1. The definitions of Function Point Analysis and the COSMIC method do not properly take into account the amount of processing required by software functional specifications.
2. If we assume –as is generally accepted– that the effort required to implement a software application is related to the number of Lines of Code to be written, the possibility of having widely different sizes in LoC for applications that have the same functional size means that functional size is not a good enough predictor of development effort.

The observation reported at point 2 above does not apply only to the coding phase. The difference in the number of classes and methods suggest that also the effort required by design and testing activities is better estimated based on measures that represent the size of the code structure –like the number of classes– rather than the functional size.

As a final remark, we can observe that McCabe complexity is similar for the two considered applications. This means that Gomoku does not need more complex code, but just more code. In other words, it is the difference in the amount of data processing, not in the complexity of the processing that is relevant, and that existing functional size fail to represent.

VI. DISCUSSION: WHAT SOLUTIONS ARE POSSIBLE?

The usefulness of the evidence given in this paper stems from a few well-known facts:

– We need to estimate, during the early phases of a project, the overall software development effort.
– Development effort has been widely reported to be directly related to the size in LoC of software. Unfortunately, the size in LoC is not available in the early phases of projects, when estimates are most needed.
– Therefore, we need FSM methods, i.e., we need measures of functional specifications, because specifications are available in the early phases of projects.
– In this paper, we provide some evidence that current FSM methods appear limited in representing the amount of data processing required by functional specifications. Therefore, we need to somehow enhance FSM methods to remove such limitation.

So, we are facing the following research question: how can we improve FSM methods so that the delivered functional size measures account for the amount of data processing described or implied by the functional specifications?

This is an open research question. Providing a final answer to it can be achieved after a substantial amount of further studies. In the following sections, we report a few observations, ideas and evaluations that could be useful considering when tackling the problem.

A. Software Models

FSM methods –like any measurement method– are applied to models of the object to be measured. Hence, a rather straightforward consideration is that data processing must be represented in the model that describes the software application to be measured.

We can observe that the conceptual model of software proposed in the COSMIC method includes data processing, but no criteria or procedures for measuring data processing are given in the context of the COSMIC method.

In COSMIC, data processing is a sub-process of a functional process. Therefore, functional processes should be described in a manner that makes it possible to identify and measure the extent of data processing that occurs within a functional process.

Given the similarity of COSMIC functional processes and FPA elementary processes (or transactions) any technique used to enhance the expressivity of COSMIC models as far as data processing is concerned should be readily applicable to FPA models as well.

B. Software Specifications

A question that should be considered is if the information required for identifying and measuring data processing is always available from the software specifications that are derived from user requirements.

Functional Size Measurement methods use models of functional specifications: if functional specifications do not include information on data processing, neither will their models, and FSM methods will not be able to account for data processing.

So, another open question is the following: is it necessary to go beyond user requirements related specifications to be able to represent data processing? In other words: should elements of design be anticipated, to get better measures of the amount of data processing to be implemented?

C. Qualitative Knowledge

Current FSM methods are inherently quantitative. Even if some measurement activities –like deciding if two sets of data should be two RET of a unique logic file or they should belong to separate logic files– involve some subjectivity, they are always meant to provide measures (the number of ILF, RET, etc.) according to ratio scales.

One could wonder if the use of more qualitative knowledge, derived through inherently subjective evaluations and expressed via ordinal scales, would more suitable for expressing the relevant information concerning data processing.

For instance, after talking with stakeholders, an analyst could easily classify the functional process “Make a move” of the Tic-tac-toe application as very simple, while the same process of the Gomoku application could be classified as very complex.

D. Towards a Measure of Data Processing

As mentioned above, proposing a solution to the problem outlined above is very difficult. Here we outline a couple of directions to be considered when addressing the problem.

A first consideration concerns the level of description of data processing. At a high level, the complexity of the processes in terms of number of different cases to be considered could easily determine the amount of data processing required. Consider for instance a process that starts by identifying users: if the specifications indicate that the user can be identified in three different ways (e.g., by name, by social security number, and by email address) it is likely that it will have to process three times as much data as a process that identifies users in a single way.

Another observation concerns how to differentiate functionalities. A possibility is to account for the internal states a function has to deal with. In the case of tic-tac-toe, the number of states in which the game can be is quite small; on the contrary, the states of a Gomoku game are very numerous. Accordingly, the amount of computation could be proportional to the number of states, since the function has to properly deal with all states. However, the quantification of data processing could be further complicated by the presence of equivalent states, i.e., sets of states that are managed in the same way, so that having N or N+1 states in such sets would not affect the amount of processing required. For instance, a data increase function has to account for months having 28, 30, or 31 days: the fact that there are 7 months having 31 days and just one having 28 days is irrelevant.

VII. Related Work

Although several FSM methods (e.g., Mark II FP, NESMA and FiSMA) have been proposed as extensions or replacements of Function Point Analysis, very little attention has been given to the measurement of data processing.

Function Point Analysis and other methods –like Use Case Points [5]– introduce a mechanism for "adjusting" the size measure to take into account additional complexity factors that are likely to increase the effort required for implementation. In fact, among FPA value adjustment factors (VAF) we find "Complex processing," which represents to what degree the application includes extensive logical or mathematical processing. This mechanism is similar to what we need, but has a few shortcomings, including:

- In FPA the considered VAF's value increases the application size by 5%: two orders of magnitude less than needed in the Tic-tac-toe vs. Gomoku case.
- The VAF applies to the whole application, so that it is not possible to distinguish simple and complex processes.

The measure of Path [6][7] represents the complexity of processes in terms of the number of execution paths that are required for each process. Although this measure proved fairly effective in improving effort estimation based on functional size measures, it is not applicable in cases like those considered in this paper, since the alternative courses of the specified processes are not known.

VIII. Conclusions

In this paper, we have shown by means of examples that functional size measurement methods fail to represent the amount of data processing required by software functional specifications.

Since we discussed just one example, one could wonder how general are the results reported in the paper. As to this issue, it is easy to see that the limits of FSM discussed in the paper apply to several programs. Consider for instance software measurement programs: from the point of view of functional size, all the measurement functions that read a set of source files and deliver a numeric value are equivalent. However, it is clear that measuring LoC is easier (i.e., it involves less data processing) than computing McCabe complexity, which in its turn is easier to compute than most coupling measures.

The work reported in the paper indicates that we need a measure that can complement Function Points or COSMIC Function Points to represent the amount of data processing that is required to provide the required functionality.

We are interested to represent and quantify the amount of data processing not because of an abstract interest in the definition of functional size measures, but because –as shown in the paper– data processing is logically related to code size, which is known to determine the amount of development effort required to build a software application.

How to measure the amount of data processing required by the specifications of a software application is an open research question of great practical interest that should receive much more attention than it currently does.

Acknowledgment

The work presented here has been partly supported by the FP7 Collaborative Project S-CASE (Grant Agreement No 610717), funded by the European Commission and by project "Metodi, tecniche e strumenti per l'analisi, l'implementazione e la valutazione di sistemi software," funded by the Università degli Studi dell'Insubria.

References

[1] A. J. Albrecht, "Measuring Application Development Productivity", Joint SHARE/ GUIDE/IBM Application Development Symposium, 1979, pp. 83-92.

[2] International Function Point Users Group. Function Point Counting Practices Manual - Release 4.3.1, January 2010.

[3] ISO/IEC 20926: 2003, Software engineering – IFPUG 4.1 Unadjusted functional size measurement method – Counting Practices Manual, Geneva: ISO, 2003.

[4] COSMIC – Common Software Measurement International Consortium, The COSMIC Functional Size Measurement Method - version 3.0.1 Measurement Manual, May 2009.

[5] G. Karner, "Resource estimation for objectory projects". Objective Systems SF AB, 17. 1993.

[6] G. Robiolo and R. Orosco, "Employing use cases to early estimate effort with simpler metrics". Innovations in Systems and Software Engineering, 4(1), 2008, pp. 31-43.

[7] L. Lavazza and G. Robiolo, "Introducing the evaluation of complexity in functional size measurement: a UML-based approach". ACM-IEEE Int. Symp. on Empirical Software Engineering and Measurement, September 2010.

[8] http://algojava.blogspot.it/2012/05/tic-tac-toe-game-swingjava.html.

[9] https://github.com/whsieh/gomoku