Measuring the Functional Size of Real-Time and Embedded Software: a Comparison of Function Point Analysis and COSMIC

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

Abstract— **The most widely used methods and tools for estimating the cost of software development require that the functional size of the program to be developed be measured, either in "traditional" Function Points or in COSMIC Function Points. The latter were proposed to solve some shortcomings of the former, including not being well suited for representing the functionality of real-time and embedded software. However, little evidence exists to support the claim that COSMIC Function Points are better suited than traditional Function Points for the measurement of real-time and embedded applications. Our goal is to compare how well the two methods can be used in functional measurement of real-time and embedded systems. We applied both measurement methods to a number of situations that occur quite often in real-time and embedded software. Our results seem to indicate that, overall, COSMIC Function Points are better suited than traditional Function Points for measuring characteristic features of real-time and embedded systems. Our results also provide practitioners with useful indications about the pros and cons of functional size measurement methods when confronted with specific features of real-time and embedded software.**

Keywords- Functional Size Measurement; Function Point Analysis; COSMIC Function Points; Real-time software; Embedded software

I. INTRODUCTION

Several methods have been proposed to estimate the development effort of a software product, given the characteristics of the product itself and its development process. Software size plays a special role in effort estimation, as it is the main input used by the vast majority of effort estimation models. Accordingly, measures of *functional* size are used in early effort estimation models, since other measures –like Lines of Code– are not available in the early development phases. Functional measures quantify the functional size of a software application, as defined in the requirements specification documents.

The available functional sizing methods are evolutions of the Function Points Analysis (FPA), originally proposed by Allan Albrecht [1]. The International Function Points User Group (IFPUG) maintains the definition of the method and publishes and regularly updates the official Function Point (FP) counting manual [2][3]. Effort estimation methods have been defined, and tools supporting them have been developed, which require the size in FP as the main input.

FP are generally not considered well suited for measuring the functional size of embedded applications. The reported motivation is that FP –conceived by Albrecht when the programs to be sized were mostly Electronic Data Processing applications– capture well the functional sizes of data storage and data movement operations, but are ill-suited for representing the complexity of control and elaboration that are typical of embedded and real-time software.

The COSMIC method was defined to overcome some limitations of FPA. The COSMIC method [4] redefines FPA's basic principles of functional size measurement in a way that applies equally well to traditional "business" application and other applications, including the real-time and embedded ones. Specifically, the COSMIC method counts the data movements (entries, exits, reads and writes) that involve data groups (corresponding approximately to FPA's logic files) in each functional process (corresponding to FPA's elementary processes). The result is a functional size measure called COSMIC Function Points (CFP).

Even though it is traditionally considered not well suited for real-time and embedded applications, FPA can be applied to embedded software via a careful interpretation of FP counting rules [5]. Moreover, it is known that many real-time projects have actually been measured using FPA. On the contrary, there is little *analytic* evidence of successful applications of the COSMIC method to real-time and embedded applications. This paper aims at providing some evidence about the suitability of FPA and the COSMIC method to measure real-time embedded software.

Both FPA and COSMIC methods require the representation of user requirements according to a method-specific model of software (e.g., the FP model includes logic files and elementary processes, while the COSMIC model includes functional processes and data movements). Measurement is then based on counting the elements of these models according to given rules. To measure RT and embedded software, it is of critical importance that representative models can be correctly derived from the user requirements. To test this ability, we consider a set of typical and representative –though necessarily incomplete– features of real-time embedded software and apply FPA and COSMIC to each of them. The comparison of the two methods provides useful indications to the developers that have to choose a functional size measurement method.

The paper is organized as follows: Section II illustrates the attractiveness of the COSMIC method from the

 ISBN: 978-1-61208-304-9

management point of view. Section III presents a set of modeling and measurement problems that occur frequently in real-time and embedded software developments. In Section IV, FPA and COSMIC methods are applied to the cases illustrated in Section III. Section V accounts for related work, while Section VI draws some conclusions and outlines future work.

Throughout the paper, we refer exclusively to Unadjusted Function Points (UFP) for FPA, because UFP are more commonly used than adjusted Function Points and because UFP are recognized as an ISO standard, while FP are not.

II. SIZING AND ESTIMATION OF REAL–TIME EMBEDDED SOFTWARE: THE MANAGER'S POINT OF VIEW

Both FPA and COSMIC methods aim at measuring the size of Functional User Requirements (FUR). However, there are a few reasons that suggest that the COSMIC method may be preferable. First, CFP are defined in a simple and sound way, while the definition of FP has been widely criticized, e.g., because the weighting mechanism make unclear whether FP are a measure of size or effort [6], or because the inherent subjectivity of FPA leads even certified measurers to measure different sizes for the same application [7][8]. Finally, the COSMIC method, which does not require a thorough analysis of data and allows for analyzing transactions at coarser granularity level, is somewhat faster and less expensive than FPA.

So, managers have a few reasons to prefer the COSMIC method over FPA. However, evidence concerning the suitability of the COSMIC method for measuring real-time software is still missing. This paper aims at filling this gap.

III. CASE STUDIES FOR FUNCTIONAL SIZE MEASUREMENT OF REAL-TIME EMBEDDED SOFTWARE

Here, we illustrate a set of typical features of real-time and embedded software that are difficult to represent by means of the models that underlie the definition of functional size measurement methods. All the proposed cases are derived from the first author's experience gained in measuring seven avionics applications in a large European company. So, the proposed set of cases is of empirical origin: during the measurement, the cases presented here emerged as those particularly challenging for functional size measurement. Most examples are illustrated by means of sequence diagrams, according to the measurement-oriented modeling methodology proposed in [9] and used in [10]. It is assumed that the reader is familiar with FPA and COSMIC concepts and terminology and with UML.

A. Embedded processes having multiple purposes

In embedded software, several processes often include both updating some data and producing some result. Consider for instance a process that initializes and tests a piece of hardware (Fig. 1): both the initialization and the test are necessary. Actually, the initialization and test of several hardware devices are performed by means of a single command: you send the initialization command and get the resulting state back, so that you can check that the device is working correctly.

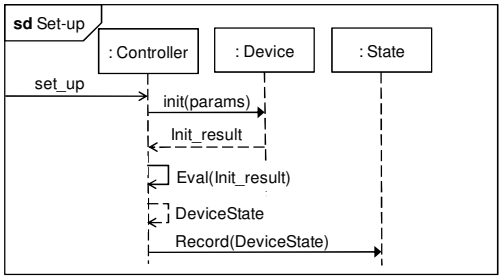

Figure 1. Inizialization of devices: the "main purpose" is not evident.

B. Transactions defined at very low level

Requirements often concern very low level operations, thus making it difficult to identify functions that match the definition of Base Functional Components.

1) Memory vs. data

In embedded software, the use of RAM as a whole introduces new requirements. For example, a piece of software embedded on board of a military airplane should clear the whole RAM under given circumstances, e.g., if the airplane crashes in an enemy zone (because the information stored in memory must not be made available to enemies). This requirement (Fig. 2) is peculiar in that it is about the whole RAM, not the user-relevant data.

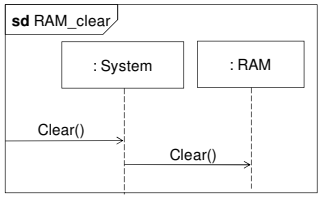

Figure 2. RAM clearing process.

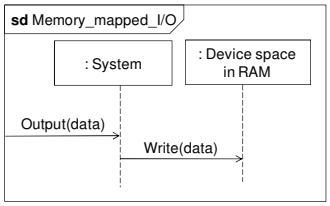

Figure 3. Memory-mapped I/O.

2) Memory mapped I/O

In embedded systems, updating a variable and sending data to a device can be extremely similar operations. For instance, when I/O is memory-mapped, both mentioned operations write registers or RAM locations (Fig. 3).

3) Processes that do not terminate properly

In embedded software, it is often required that a function terminates by jumping to a given location. This situation is illustrated in Fig. 4: the initialization function terminates by executing the set-up function (described in Fig. 10).

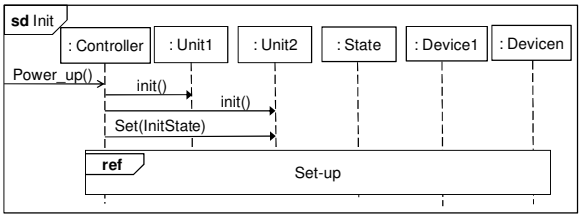

Figure 4. A function that ends with a jump to another function.

C. Taking into account the devices

In traditional software applications, functions are usually invoked by the user and end either by updating some internal data, or by outputting some information. In embedded applications, the situation can be very different. Often it is some hardware device (not a user) that acts as both the cause

that determines the execution of the function and the destination of the produced data or signals.

1) Considering the role of the Operating System in I/O

Let us consider the following requirements for an I/O functionality (described in Fig. 5): "upon request by the controller, data are retrieved from an I/O channel, according to the criteria stored in the I/O channel table. When all the data have been read, they are suitably converted and sent back to the controller." It is often the case that the I/O operation has to be carried out with the help of the Operating System and the requirements can be implemented by means of two functions, illustrated in Fig. 6 and Fig. 7. The first function (Fig. 6) is invoked by the controller and prepares an I/O request for the OS and a subsequent system call. The second function (Fig. 7) is triggered by the interrupt from the I/O device and involves reading the data from the channel, elaborating them, and sending them back to the controller. The execution of this "function" is done partly by the OS (by a driver that will have to be implemented as a part of the application development) and partly in the section of the application devoted to I/O.

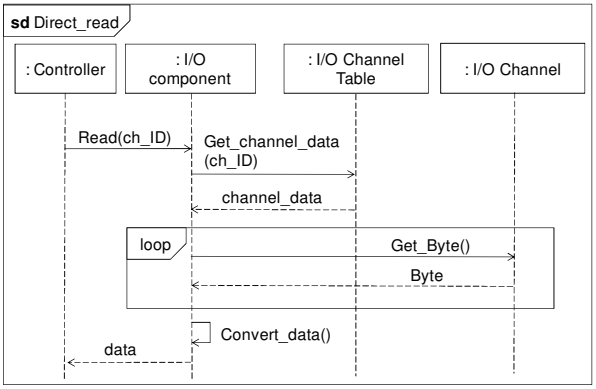

Figure 5. Process featuring direct access to I/O channels.

If the development also includes the construction of a driver for the considered I/O device, it seems that taking into account the size of the corresponding code will contribute to produce a more accurate effort estimate. In other words, it seems reasonable to count two functions, corresponding to the "elementary processes" described in Fig. 6 and Fig. 7.

2) Multi cycle operations

In real-time systems, it is not unusual that a function is too long to fit into one execution cycle. In such cases, it is rather common to split the function into two (or more) pieces that are executed in consecutive execution cycles. Here are two typical examples:

- The function transfers data via a buffer. The data to be transferred do not fit in the buffer. The transfer is split into n cycles: in each cycle 1/n of the data are copied into the buffer.
- The function, triggered by the tick, takes a time longer than the cycle duration (i.e., the time between two consecutive ticks) to execute. Thus, the transfer is split into multiple consecutive cycles.

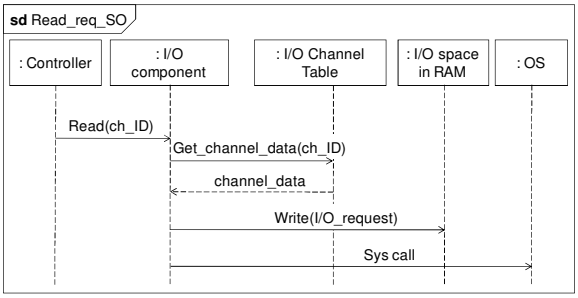

Figure 6. Process Access to I/O channels via the O.S.

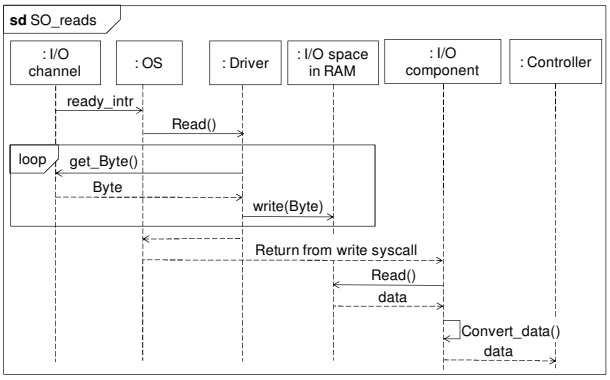

Figure 7. The O.S. handles the I/O.

An example is given in Fig. 8: an output operation is split over two consecutive clock cycles. In the first cycle the application outputs the data from Data_1 and sets the State to represent that there is a pending output operation; in the following cycle, the State indicates that the output operation has to be completed, thus data are read from Data_2 and sent to the output device.

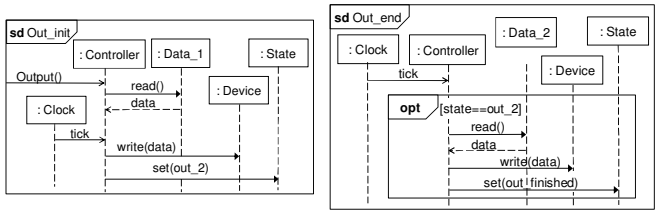

Figure 8. Output: first and second (final) cycle.

These cases are often described in the requirements, since they deal with the real-time behavior of the application, which is typically explicitly accounted for in the requirements specification.

However, requirements specifications could not state explicitly that the function should be split, i.e., requirements could just describe the whole operation as in Fig. 9.

D. Long processes

In embedded software, functions are often "service routines" that perform rather long tasks; e.g., the requirements specify that "the connected devices are tested, and the result (a 'pass' value or the set of diagnostics) is sent to the controller, which stores it for later use." Fig. 10 illustrates the situation with 4 different device types.

 ISBN: 978-1-61208-304-9

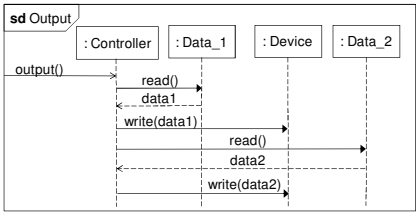

Figure 9. Output, not split.

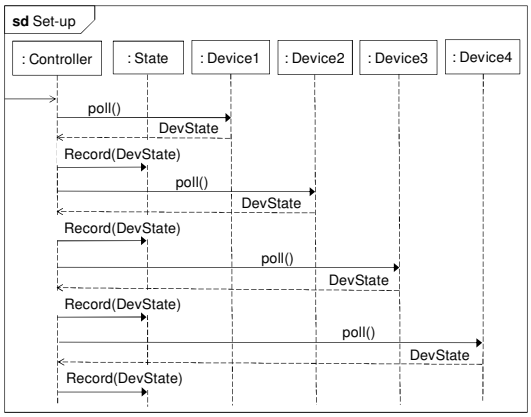

Figure 10. A long transaction.

E. Unusual data

Embedded applications often include constant data structures (e.g., data mapping tables or bit masks) that require a non-negligible design effort, which we would like to take into account. An example is shown in Fig. 5: for each request to read an I/O channel, the I/O component reads from the channel table how many bytes must be read from the channel and how they should be interpreted. The channel table is a read-only structure that describes how to manage the I/O channels.

F. Complex elaborations

In real-time and embedded applications, some operations can be complex. Consider for instance the generic flight control operations described in Fig. 11. It should not be surprising that the computation of the flight control data can be quite complex.

IV. Applying FPA and COSMIC to Real-Time Embedded Software

This section illustrates the application of FPA and COSMIC methods to the cases described in Section III.

A. Embedded processes having multiple purposes

According to the IFPUG counting rules [2][3], the size of a function varies according to its type (external input, output or query). The type is determined by the "main purpose" of the function, according to the requirements. However, it may be difficult to decide what the main purpose is, since both the external input and the external output can update internal data and report a result, as in our case. In conclusion, measures based on FPA have some degree of subjectivity that can be hardly avoided.

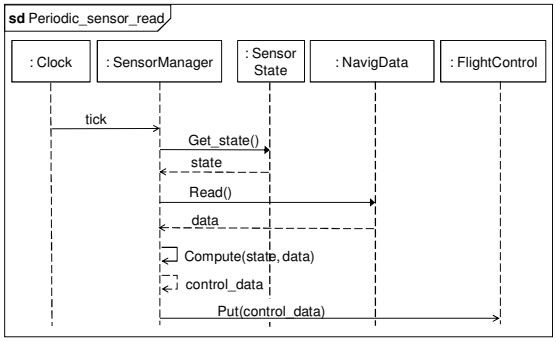

Figure 11. Sensor-driven flight control.

The problem described above does not apply to COSMIC measurement, since all processes are treated in the same way, regardless of their purpose.

B. Transactions defined at very low level

1) Memory vs. data

According to the principles of FPA, in a case like the one described in Section III.B.1) one should count the memory clearing function as an external input. In that case, since every External Input (EI) manages an Internal Logic File (ILF), we should consider the RAM an ILF. On the one hand, counting the RAM as an ILF does not appear correct with respect to the rules, since logic data files should represent a homogeneous set of related data (which RAM is not), on the other hand, not considering the RAM as an ILF is an inconsistency, as all EI have to deal with an ILF.

There is a similar problem with the COSMIC method, as the process writes in the RAM: accordingly, we should consider a write data movement. However, this implies that the RAM is classified as a data group, which does not appear perfectly coherent with the COSMIC rules.

2) Memory mapped I/O

When I/O is memory-mapped, an output operation can be modeled as an External Output (EO) but also as an EI since the output is obtained by writing registers or RAM locations (see Fig. 3). The choice affects the resulting measure, since EI and EO have different weights. With the COSMIC method, you still can model the operation as a Write or an Exit data movement, but the choice does not affect the final measure, since every data movement contributes exactly one CFP.

3) Processes that do not finish properly

According to FPA, a transaction function has to be self-contained and leave the application being counted in a consistent state. In embedded software, it is often required that a function terminates by jumping to a given location (Fig. 4). In this case, the transaction is not self-contained and does not leave the program in a consistent state. FPA does not suggest how to take into consideration this type of functions. Just ignoring them would not be a good idea, since it takes some effort to implement these functions; hence we want them to contribute to the functional size of the application. Actually, there is no other way of dealing with these cases than just ignoring the constraints imposed by the IFPUG and counting the functions, considering their

 ISBN: 978-1-61208-304-9

behavior down to the final jump. The same problem occurs when the COSMIC method is used, since functional processes are defined as FPA transactions, in essence.

C. Taking into account the devices

1) Considering the role of the Operating System in I/O

With both FPA and COSMIC methods, the measurement of the process represented in Fig. 5 is quite straightforward. The problem here occurs when the development must also include the construction of a driver for the considered I/O device, since taking into account the size of the corresponding code will contribute to produce a more accurate effort estimate. In other words, it seems reasonable to count two functions, described in Fig. 6 and Fig. 7.

With FPA, this requires a deviation from the FPA counting practice, since FPA does not take into account the existence of different "layers": with FPA you can only measure requirements at the single abstraction level corresponding to the user's point of view, and the user is not aware of the OS and what happens in the OS.

With the COSMIC method, it is possible to explicitly model and measure the layers that compose the software application. The sum of the sizes of the layers is generally greater than the size of the whole application corresponding to the point of view of the user (who is not aware of the existence of layers). So, the measure of layers is exactly what is needed to take into account the size of the OS parts that are being developed.

2) Multi cycle operations

The cases described in Section III.C.2) suggest that the value of a functional size measure can depend on how requirements are written. Let us consider the case when requirements specifications do not state explicitly that the function should be split (Fig. 9): if Data_1 and Data_2 account for 10 DET each, the transaction is a high complexity EO (having 3 FTR and 21 DET), whose size is 7 FP. When requirements specifications prescribe that the function be split (Fig. 8) we have two average complexity EO (3 FTR and around 12 DET each), whose size is 10 FP in total. When requirements specifications do not state explicitly that the function should be split, the COSMIC method identifies one functional process sized 5 CFP, since it involves 5 data movements (the Entry, the Reads of Data_1 and Data_2, and the corresponding Exits). When requirements specifications prescribe that the function be split, according to the COSMIC rules we have two functional processes, one involving 5 data movements (the Entry that triggers the operation, the Read of Data_1, the Entry of the clock tick, the Exit to the device, the Write of the state), and one involving 4 data movements (the Entry of the tick, the Read of Data_2, the Exit to the device, the Write of the state); the total size is thus 9 CFP.

In conclusion, both methods provide measures of size that depend on how requirements are written. This is a characteristic of the methods that has to be taken into account, as it affects the resulting measures.

D. Long processes

A well known problem with Function Points is the so-called "cut-off" effect: a function cannot contribute more than 7 FP to the functional size, regardless how many DETs it moves and how many FTRs it involves. This is a relevant problem, especially in embedded software, where functions are often "service routines" that perform rather long tasks, like in the example illustrated in Section III.D and Fig. 10.

Fig. 10 illustrates the situation with 4 different device types. According to the IFPUG counting rules, this is a single transaction. If the device states contain on average 5 (or more) parameters, then the transaction is a complex one. The problem here is that if we had 5 or more different types of devices, the number of FP would not increase with the number of devices: according to FPA, we would have just one complex EI. This is a problem, because in practice the development effort increases with the number of device types, since each device type provides different status data, which need to be interpreted in a specific way.

FPA hides from the estimation methods how much a function is bigger (thus more expensive to build) than another that classifies as complex. The COSMIC method, on the contrary, does not suffer from the cut-off effect. In a case like the one in Section III.D and Fig. 10, the size in CFP takes into account *all* the data movement, whose number is proportional to the number of devices.

E. Unusual data

According to FPA, data functions are either internal data "maintained" (i.e., modified) by the application, or external data (maintained outside the application). Constant data are treated as "decoding data" and explicitly excluded from the counting [2]. However, it seems that the authors of the IFPUG manual had in mind simple "zero effort" constants when they wrote the rules concerning the constant data.

To account for the fact that a constant data structure will require some design effort, it is necessary to deviate from the IFPUG rules, and count a "constant ILF": for instance, in the example illustrated in Fig. 6, one should count an ILF for the channel table; consistently, a FTR for each access to the table should be considered.

The COSMIC method does not count data directly; that is, no fraction of the size measures accounts for data. On the contrary, data movements are counted without considering whether the data being moved are constant or not. In conclusion, this case does not pose any additional difficulty to the application of the COSMIC method.

F. Complex elaborations

Both FPA and COSMIC methods base the measurement of size on the number of processes and the amount of data handled. For instance, the process described in Fig. 11 is considered as an EO (with a maximum size of 7 FP) or a functional process accounting for 4 CFP (as it involves 4 data movements). None of the two methods considers the complexity of the computations performed: the fact that the "Compute" operation performed in the process is simple or complex does not change the size of the process.

 ISBN: 978-1-61208-304-9

This is clearly a shortcoming of the two methods, since the development effort is very likely proportional to the complexity of the functions to be implemented.

V. RELATED WORK

There is a fairly large body of literature aimed at extending the scope of functional size measurement to real-time software. Mark II Function Points [11][12] refine and extend the traditional function point transaction model and environmental factors. Asset-R [13] extends the applicability of FP to real-time systems by considering issues like concurrency, synchronization, and reuse. It also accounts for architectural, language expansion, and technology factors to generate the size estimate. Application Features [14] aim at the early estimation of the size of application in the process control domain. Counting practices for highly constrained systems [15] address issues such as boundary identification and internal processing. Also the IFPUG published a Case Study on how to apply FPA to real-time software [16].

A common characteristic of the methods mentioned above is that none of them is widely used in practice. A partial exception is represented by Mark II Function Points [11], which were also standardized [12]. So, the popularity of FPA and COSMIC suggested that their suitability to deal with software has to be evaluated.

VI. CONCLUSIONS

The results of our analysis show (see Table I) that several cases can be measured with the COSMIC method by just applying the measurement rules given in the manual [4], while Function Point Analysis often requires "bending" the rules to account for the considered cases. Also the resulting measures are easily affected by the measurement choices made in FPA, while there are just a few cases (namely, processes terminating with a jump, multi-cycle operations and complex elaborations) that can affect the measures in CFP.

TABLE I. COMPARISON OF FSM METHODS

Case	FPA		COSMIC	
	Rules	*Meas.*	*Rules*	*Meas.*
Multiple purpose processes	✗	✗	✓	✓
Memory data	✗	✗	✓	✓
Memory mapped I/O	✗	✗	✗	✓
Processes terminating with jump	✗	✗	✗	✗
Clock	✓	✓	✓	✓
OS involved in I/O	✗	✗	✓	✓
Multi cycle operations	✓	✗[a]	✓	✗[a]
Long processes	✗	✗	✓	✓
Unusual data	✗	✗	✓	✓
Complex elaborations	✗[b]	✗	✗	✗[b]

[a] The measures depend on how requirements are written.

[b] Elaboration complexity is just not accounted for by any rule.

In conclusion, the original claims that the COSMIC method is more suitable than FPA for measuring real-time and embedded applications seem justified.

In any case, it must be noted that neither FPA nor the COSMIC method account for the complexity of the required elaboration. This may be a problem in the real-time embedded context, since some processes can be really very complex and require a relevant amount of development effort. Future work involves assessing measures that represent not only the functional size of Real-Time applications as done by FPA and COSMIC methods, but can represent also the complexity of the required elaboration.

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] L. Lavazza and C. Garavaglia, "Using Function Points to Measure and Estimate Real-Time and Embedded Software: Experiences and Guidelines", ESEM 2009, Lake Buena Vista, FL, USA, October 15-16, 2009, IEEE, pp. 100-110.

[6] A. Abran and P.N. Robillard "Function points: a study of their measurement processes and scale transformations", Journal of Systems and Software, vol.25,n.2, Elsevier, 1994, pp.171-184.

[7] C. Kemerer, "Reliability of Function Points Measurement: a Field Experiment," Comm. ACM, Vol. 36, No. 2, 1993, pp. 85-97.

[8] J.R. Jeffery, G.C. Low, and M.A Barnes, "Comparison of Function Point Counting Techniques," IEEE Trans. Software Eng., Vol. 19, No. 5, 1993, pp. 529-532.

[9] L. Lavazza, V. del Bianco, and C. Garavaglia, "Model-based Functional Size Measurement", 2nd Int. Symp. on Empirical Software Engineering and Measurement – ESEM 2008, Kaiserslautern, Germany. October 9-10, 2008, pp. 100-109.

[10] L. Lavazza and V. del Bianco, "A Case Study in COSMIC Functional Size Measurement: the Rice Cooker Revisited", IWSM 2009, Amsterdam, November 2009, pp. 101-121.

[11] C.R. Symons, "Function Point Analysis: Difficulties and Improvements", IEEE Transactions on Software Engineering, Vol. 14, No. 1, January, 1988, pp. 2-11.

[12] ISO/IEC 20968: 2002, Software engineering Mk II Function Point Analysis. Counting Practices Manual, International Standardization Organization, ISO, Genève, 2002.

[13] D. J. Reifer, "Asset-R: A Function Point Sizing Tool for Scientific and Real-Time Systems", Journal of Systems and Software, Vol. 11, No. 3, March 1990, pp. 159-171.

[14] T. Mukhopadhyay and S. Kekre, "Software Effort Models for Early Estimation of Process Control Applications", IEEE Transactions on Software Engineering, Vol. 18, No. 10, October 1992, pp. 915-924.

[15] European Function Point Users Group, Function Point Counting Practices for Highly Constrained Systems, 1993.

[16] IFPUG, Case Study 4: Counts Function Points for a Traffic Control System with Real Time Components, International Function Point Users Group – IFPUG.

 ISBN: 978-1-61208-304-9