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

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— **Function Points Analysis and the COSMIC method are very often used for measuring the functional size of programs. The COSMIC method was proposed to solve some shortcomings of Function Points, 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. To help practitioner choose a method for measuring real-time or embedded software, some evidence of the merits and shortcomings of the two methods is needed. Accordingly, our goal is to compare how well the two methods can be used in the functional measurement of real-time and embedded systems. To this end, we applied both measurement methods to the situations that occur quite often in real-time and embedded software and are not considered by standard measurement practices. Our results indicate that, overall, COSMIC Function Points are better suited than traditional Function Points for measuring characteristic features of real-time and embedded systems.**

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 need for software development estimation and size measurement applies to RT software as well [1].

The available functional sizing methods are evolutions of Function Points Analysis (FPA), originally proposed by Allan Albrecht [2]. 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 [3][4]. Effort estimation methods have been defined, and tools supporting them have been developed, which require the size in FP as the main input [5].

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 [6] redefines FPA's basic principles of functional size measurement in a way that applies equally well to traditional "business" application and other applications, including 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 [7]. 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 real-time 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 somewhat extensive –though necessarily incomplete– set of typical and representative 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.

Even though both FPA and COSMIC methods aim at measuring the size of Functional User Requirements (FUR), 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 [8], or because the inherent subjectivity of FPA leads even certified measurers to provide different size measures for the same application [9][10]. 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.

In this paper, we enhance the work reported in [1] by considering additional characteristics of real-time and embedded software –namely the usage of clocks and timers– that could make the application of functional size measurement rules challenging. Consequently, the discussion on the comparison of FP and CFP is extended to the newly considered cases. In addition, the section on related work was also extended.

This paper is of interest to researchers and practitioners who are familiar with the usage of Function Point Analysis and the COSMIC method for measuring traditional software applications. Accordingly, we take for granted that the readers are familiar with the concepts and terms of functional size measurement in general and FPA and the COSMIC method in particular. Readers who are not familiar with Functional Size Measurement methods can have a look at a concise introduction to FPA and COSMIC [29] and at a glossary of metrics terms [30].

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 [4], while FP are not.

The paper is organized as follows: Section II accounts for related work. 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, while Section V draws some conclusions and outlines future work.

II. RELATED WORK

There is a fairly large body of literature aimed at extending the scope of functional size measurement to software applications that do not belong to the Information Systems domain, for which FPA was originally conceived by Albrecht. An overview of these proposals (rather old but still relevant) can be found in [13]. Among the notable attempts to adapt FPA to real-time software are:

– Feature Points [14], which include an algorithmic element and define new environmental complexity factors.
– Mark II Function Points [15][16], which refine and extend the traditional function point transaction model and environmental factors.
– Asset-R [17], which extends the applicability of function points to real-time systems by considering issues like concurrence, synchronization, and reuse. It also accounts for architectural, language expansion, and technology factors to generate the size estimate.
– 3D Function Points [18], which consider three dimensions of the application to be measured: Data, Function, and Control. The Function measurement considers the complexity of algorithms; and the Control portion measures the number of major state transitions within the application.
– Application Features [19], which aims at the early estimation of the size of application in the process control domain.
– Counting practices for highly constrained systems [20], which address issues such as boundary identification and internal processing.

Also the IFPUG published a Case Study that shows how to apply FPA to real-time software [21].

Another set of proposed approaches to make FP measurement applicable to real-time software took into account the object-oriented programming paradigm. Actually, these approaches address every type of object-oriented program or model, including real-time and control applications.

Object Points [22] are an object-oriented approach that measures the external, internal, application and object size of object-oriented systems. Objects are viewed as mini applications where each object encapsulates data and operations. A simple mapping is established between object operations (services) to transactions, and object data (attributes) to ILFs.

Class points [23] are based on the number of services required (NSR), the number of external methods (NEM) and the number of attributes (NOA) of classes. The complexity of a class is evaluated on the basis of its NSR, NEM and NOA, and then classes are weighed according to their complexity and type. Class types are: Problem Domain, Human Interaction, Data Management, Task Management. The sum of the weights gives the number of class points.

Object-oriented FPs [24] are computed following the function point counting procedure. Classes within the application boundary correspond to ILFs, while classes outside the application boundary (including libraries) correspond to EIFs. Inputs, Outputs and Inquiries are all treated in the same way: they are called generically "service requests" and correspond to class methods. The complexity of ILFs and EIFs depends on the number and type of attributes and associations. The complexity of service requests depends on the number and type of method parameters. Several ways of considering class aggregates and generalization hierarchies are proposed, thus the measured size depends on the criterion used to consider class aggregation and generalization.

Among the proposed approaches –none of which seems to have succeeded in gaining market acceptance– Full

Function Points (FFP) are quite relevant. FFP [25] take into account the differences between traditional applications and real-time applications by extending the FPA by means of new data and transactional function types:

– Updated control group: a group of data –used by the application to control, directly or indirectly, the behavior of an application or of a mechanical device– updated by the application being measured.
– Read-only control group: A group of control data used, but not updated, by the application being counted.
– Entry / Exit: A sub-process that receives / sends control data across the application's boundary.
– Read / Write: A sub-process that reads / writes a group of control data.

Also FFP did not prove successful in dealing with the functional size measurement of real-time software; hence, their authors decided to thoroughly review their definition, thus arriving at the definition of COSMIC function points [6][26]. CFP retain the basic principles of functional size measurement as in FPA, but they are defined in a manner that applies equally well to traditional "business" application and to other applications, including the real-time and embedded ones.

Several papers have been written on the suitability of Full Function Point and COSMIC Function Points to measure real-time software.

Desharnais and Morris stress the possibility of identifying and measuring different layers with the COSMIC method, and dealing with the "cut-off" effect we discussed in Section IV. IV.D [27].

Oligny et al. report about the applicability of FFP in general, based on the experiences gained while measuring seven projects (four of which real-time) [28]. The discussion does not address any of the details reported in our paper.

In conclusion, the literature is relatively rich in proposals for extending or adapting existing functional size measurement methods to real-time and embedded software; however, none of such proposals appears to be widely used in practice (possibly with the partial exception of Mark II Function Points [15], which were also standardized [16]).

So, the popularity of FPA and COSMIC suggested that they are candidates for real-time and embedded software measurement. However, nobody –to the best of our knowledge– investigated the actual applicability of IFPUG and COSMIC measurement rules to real-time and embedded software.

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 of the cases shown here 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. Even though the cases considered here were all derived from the avionic domain, they were observed in quite different applications. Accordingly, we believe that the cases presented here are representative of the challenging cases that can occur when measuring real-time and embedded software applications.

Most examples are illustrated by means of sequence diagrams, according to the measurement-oriented modeling methodology proposed in [11] and used in [12]. It is assumed that the reader is familiar 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). The initialization and test of several hardware devices are performed by means of a single command: the initialization command is sent to the devices and the resulting state is sent back, so that it is possible to check whether the device is working correctly.

In these cases, the initialization and the test are both necessary and equally important.

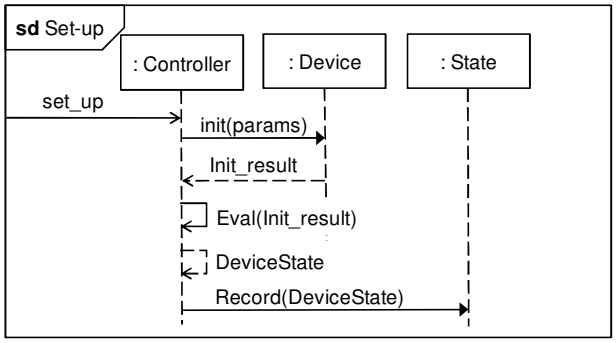

Figure 1. Inizialization of devices: the "main purpose" is not evident.

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.

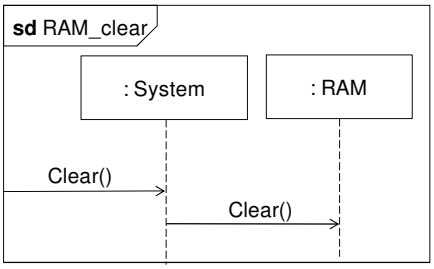

Figure 2. RAM clearing process.

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 about some specific user-relevant piece of data.

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).

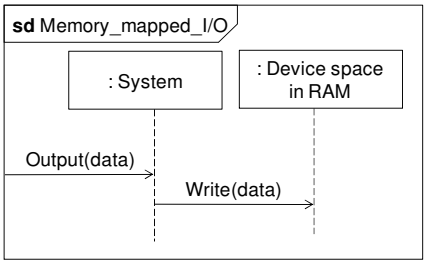

Figure 3. Memory-mapped I/O.

3) Processes that do not terminate properly

In embedded software, it is often required that a function terminate 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. 17).

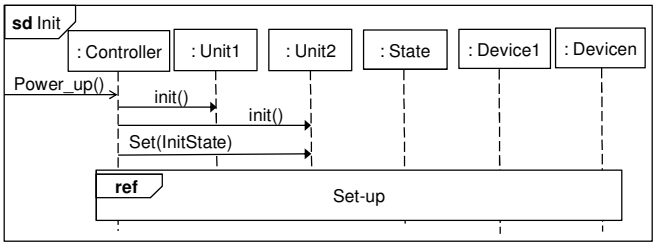

Figure 4. A function that ends with a jump to another function.

C. Taking into account the devices

In traditional software applications, functions are usually invoked by the user and end by either updating some internal data, or 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.

For instance, functional processes are often initiated by clocks and timers.

1) The Clock

Some functions are triggered periodically by clock signals. In such cases, the clock acts as a user that invokes a function: in fact, the resulting behavior is the same obtained by a human user that periodically invokes the function.

This situation is illustrated in Fig. 5: in this example, an aircraft is equipped with sensors, which collect navigation data, and a clock periodically invokes a sensor manager that asks navigation data from the sensors and sends the returned data to the flight control unit.

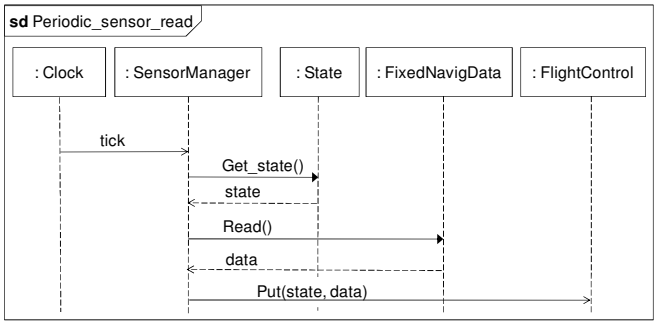

Figure 5. An elementary function triggered by the clock.

2) Timers

In embedded systems, functions can be triggered by timers, or they can terminate after programming a timer. Consider the following specifications:

Spec. A (Fig. 6): "The program sends a request for data to device X, then reads the data sent by X and stores them for later use."

Spec. B (Fig. 7): "The program sends a request for data to device X, waits for 10 ms, then reads the data from X and stores them for later use."

Spec. C (Fig. 8): "The program sends a request for data to device X; then, every 10 ms the program checks whether the data from X are ready: if so it stores them for later use."

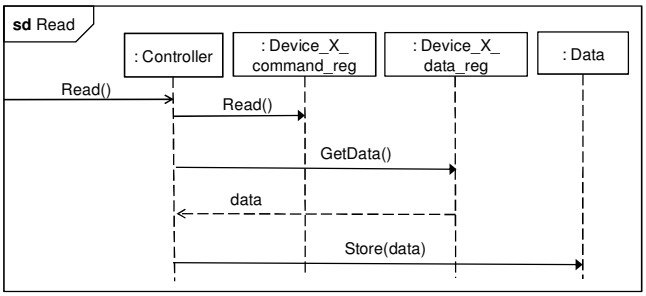

Figure 6. Device read specifications not mentioning time.

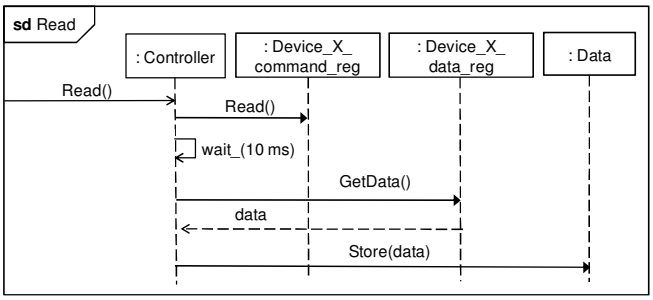

Figure 7. Device read specifications not mentioning time delay.

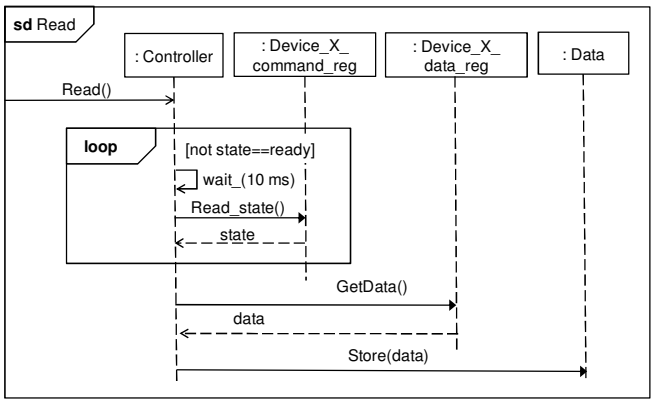

Figure 8. Device polling specifications.

All of the above specifications could be rewritten by explicitly mentioning the use of timers. For instance, Spec. C could be rewritten as follows:

Spec. D: "The program sends a request for data to device X and programs a 10 ms timeout; upon receiving the timeout signal, the program checks whether the data from X are ready: if so it stores them for later use and disables the timer, otherwise, a new 10 ms timeout is programmed."

The first part of this specification is described in Fig. 9, while the second part is described in Fig. 10

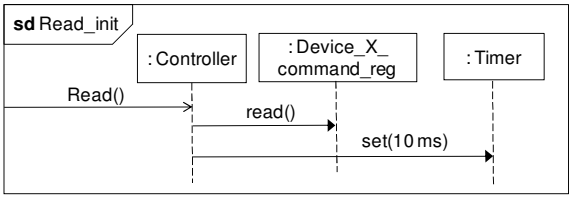

Figure 9. A transaction that programs a timer.

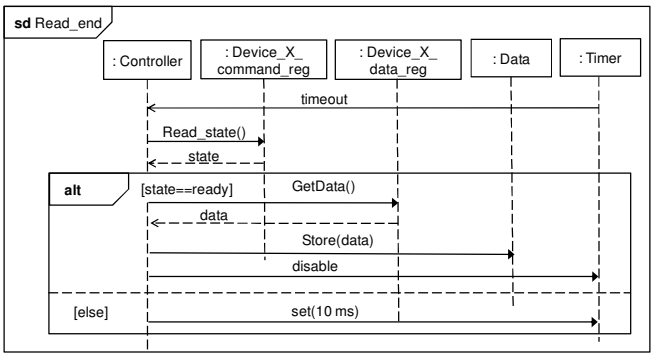

Figure 10. The timer triggers the conclusion of the operation.

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

Let us consider the following requirements for an I/O functionality (described in Fig. 11): "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 Figs. 12 and 13. The first function (Fig. 12) is invoked by the controller and prepares an I/O request for the OS and a subsequent system call. The second function (Fig. 13) 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.

If the development also includes the construction of a driver for the considered I/O device, taking into account the size of the corresponding code will contribute to produce a more accurate effort estimate. In other words, it appears reasonable to count two functions, corresponding to the "elementary processes" described in Figs. 12 and 13.

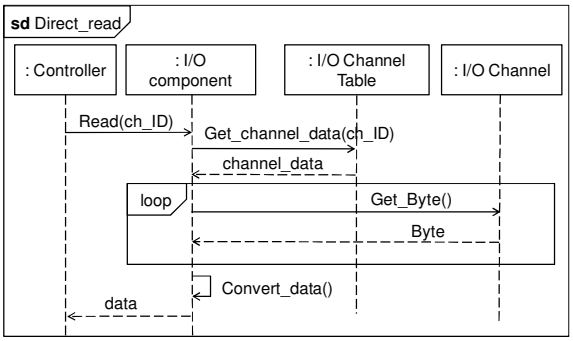

Figure 11. Process featuring direct access to I/O channels.

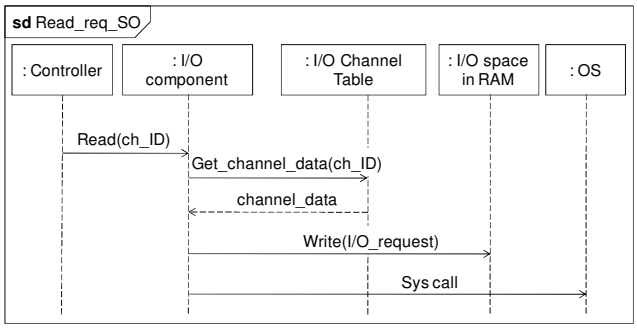

Figure 12. Process Access to I/O channels via the O.S.

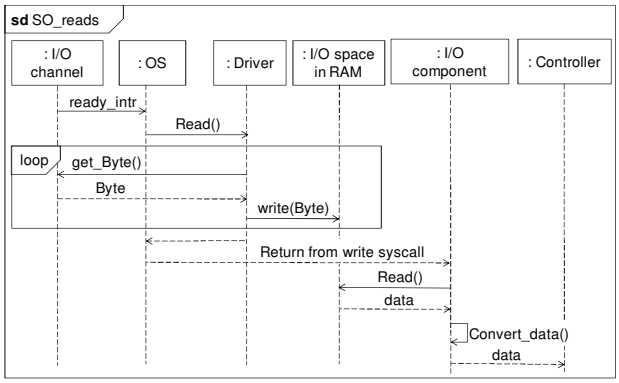

Figure 13. The O.S. handles the I/O.

4) 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.

An example is given in Figs. 14 and 15: an output operation is split over two consecutive clock cycles. In the first cycle (Fig. 14), 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 (Fig. 15), the State indicates that the output operation must be completed, thus data are read from Data_2 and sent to the output device.

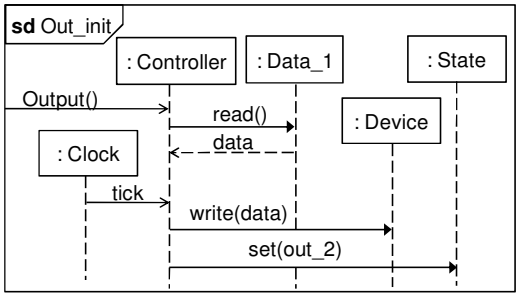

Figure 14. Output: first cycle.

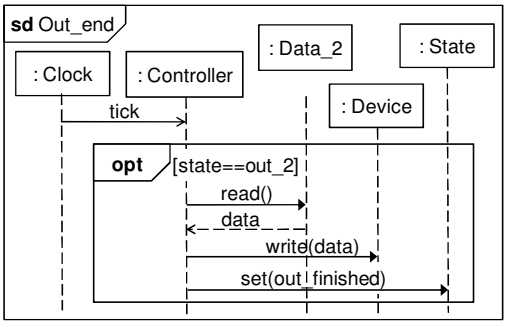

Figure 15. Output: second (final) cycle.

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. 16.

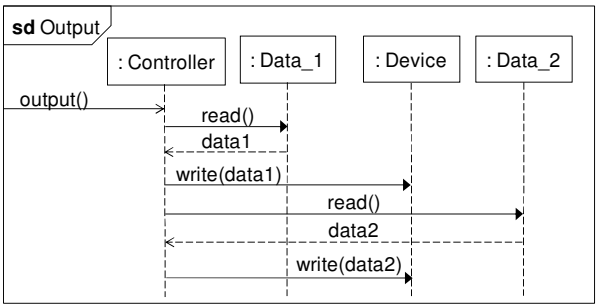

Figure 16. Output, not split.

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. 17 illustrates the situation with 4 different device types.

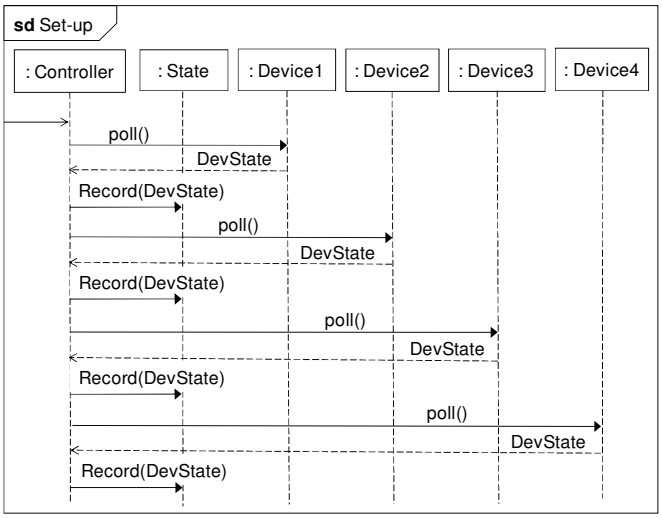

Figure 17. A long transaction.

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. 12: 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.

With respect to other elements of the system, the channel table differs only in that it is read-only; apart from that, it concerns information that is relevant to the user and it must be properly designed to be effectively and efficiently read.

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. 18: it is quite likely that the computation of the flight control data is rather complex.

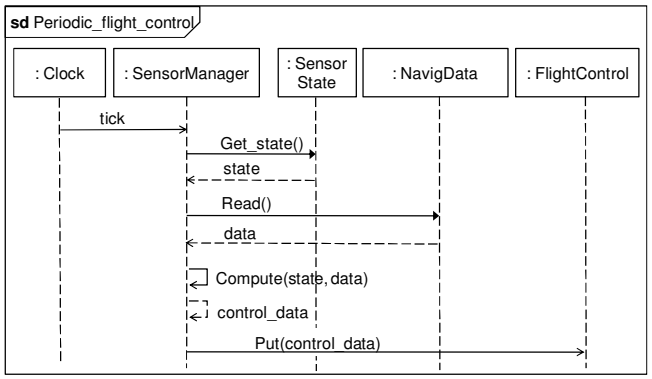

Figure 18. Sensor-driven flight control.

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 [3][4], 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.

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 Logical 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.

In the COSMIC method, there is no rule that clearly prevents treating the RAM as the object of interest involved in the process that clears the RAM. Accordingly, we would have a process involving a write data movement.

2) Memory mapped I/O

When I/O is memory-mapped, an output operation can be modeled as an External Output (EO) in FPA 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 deal with 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 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) The Clock

In functions that are triggered periodically by clock signals (as in Fig, 5), the clock acts as the user that invokes the function.

From a COSMIC point-of-view, the clock is the functional user that generates the triggering Entry (i.e., a message that informs the software that the functional user is initiating a functional process). The possibility that a device acts as a functional user is explicitly stated in the COSMIC counting manual [6]. Accordingly, the functional process illustrated in Fig. 5 involves the following data movements:

- The entry of the clock tick;
- Reading the current state;
- The request to for navigation data (an exit);
- The entry of navigation data;
- The output of state;
- The output of navigation data.

The sensor is another functional user. For this software, an event occurs when it is time to update the navigation data: the clock triggers the software, by sending it a message (triggering Entry).

The considerations reported above can be applied to FPA as well. Thus, we shall simply consider the clock as a user that can originate the execution of a function. The rest of the measurement is carried out easily according to FPA rules [3].

2) Timers

Let us consider Spec. A (Fig. 6): in this specification there is no mention of the time delay between the request to read and the retrieval of data. According to FPA, the specification involves a single transaction (an external input, if the main intent is storing the data retrieved from the sensor). Similarly, it is a single COSMIC functional process.

Specifications B (Fig. 7) and C (Fig. 8) are functionally equivalent to Spec. A, with the only difference that they mention time. By the way, Spec. B and C are also similar to each other, the only difference being that, in Spec. B, we are sure that, 10 ms after issuing the read command, the required data are ready, while in Spec. C this is not true. However, as with Spec. A, we have just one transition according to FPA, or a functional process according to the COSMIC method.

Specifications B and C (and possibly A as well) can be implemented with or without using a timer: in fact the 10 ms waiting could be achieved via a sort of busy loop. Of course, the functional size (either in FP or in CFP) of the specifications does not depend on how it will be implemented.

The problem is that the same operation described by Specifications B and C could be described as in Spec. D (Fig. 9 and Fig. 10). In this case, the analyst is just specifying delays by means of timers, which –by the way– are the most obvious means to implement the operation. As a consequence, we have two FPA transactions: the one that initialized the device (described in Fig. 9) and the one that reacts to timeout signals (Fig. 10).

In practice, specifications B and C are both low complexity External Input transactions (under the hypothesis that the data read from the sensor includes a small set of DET), accounting for 3 FP.

From a COSMIC point-of-view, Spec. B involves 5 data movements (the Entry of the triggering event, the Exit of the Read command, the Exit of the GetData command, the Entry of data, the Write of data), while Spec. C involves 6 data movements (the Entry of the triggering event, the Exit of the Read_state command, the Entry of the state, the Exit of the GetData command, the Entry of the data, the Write of data).

Spec. D involves a low complexity External Output transaction (Fig. 9) and a low complexity External Input transaction (Fig. 10): therefore, it has a size of 3+4=7 FP. From a COSMIC point-of-view, Spec. B involves a functional processes (Fig. 9) comprising 3 data movements, and a functional process (Fig. 10) comprising 7 data movements: therefore, it has a size of 3+7=10 CFP.

In conclusion, the problem here is that mentioning the timers in the specifications (which is quite natural for real-time software analysts) causes the functional size of the specified operations to increase substantially.

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

With both FPA and COSMIC methods, the measurement of the process represented in Fig. 11 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 producing a more accurate effort estimate. In other words, it appears reasonable to count two functions, described in Figs. 12 and 13.

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.

4) Multi cycle operations

The cases described in Section III.C.4) 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. 16): 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. 14 and 15) 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. 17.

Fig. 17 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 bigger a function is (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. 17, 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 [3]. 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. 12, 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. 18 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.

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. CONCLUSIONS

The results of our analysis (summarized in Table I) show that some situations that are typical of real-time and embedded applications make it necessary to interpret or "bend" the rules provided by official measurement manuals [3][6]. However, this happens more often for IFPUG Function Point Analysis than with the COSMIC method.

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, explicitly mentioned timers and complex elaborations) that can affect the measures in CFP.

TABLE I. COMPARISON OF FSM METHODS

Case	FPA		COSMIC	
	Easy application of rules	*Measure affected*	*Easy application of rules*	*Measure affected.*
Multiple purpose processes	×[a]	×	✓	✓
Memory data	×	×	✓	✓
Memory mapped I/O	×	×	×	✓
Processes terminating with jump	×	×	×	×
Clock	✓	✓	✓	✓
Timers	✓	×[b]	✓	×[b]
OS involved in I/O	×	×	✓	✓
Multi cycle operations	✓	×[b]	✓	×[b]
Long processes	×	×	✓	✓
Unusual data	×	×	✓	✓
Complex elaborations	×[c]	×	×[c]	×

[a] The application of the rule is subjective.

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

[c] 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 appear justified. The cases that were used to evaluate the applicability of FPA and COSMIC to real-time and embedded software are sort of application-independent patterns: accordingly, the results reported here are expected to be applicable to a wide range of real-time and embedded software applications.

A straightforward consequence of the study reported here is that the COSMIC method is more suited for the functional measurement of real-time software; however, considering that functional size measures are often used for effort estimation, a problem could arise with the availability of CFP-based models of real-time software development effort. To derive such models, historical data are needed. The study reported here could also provide hints for converting FP measures into CFP measures, thus obtaining the datasets that are necessary to derive effort models.

In any case, 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.

ACKNOWLEDGMENT

The work reported here was 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] L. Lavazza and S. Morasca, "Measuring the Functional Size of Real-Time and Embedded Software: a Comparison of Function Point Analysis and COSMIC", 8th Int. Conf. on Software Engineering Advances – ICSEA 2013, October 27 - November 1, 2013 - Venice, Italy

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

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

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

[5] J. E. Matson, B. E. Barrett, and J M. Mellichamp, "Software development cost estimation using function points,", IEEE Transactions on Software Engineering, vol.20, no.4, Apr 1994, pp.275-287.

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

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] T. Hastings, "Adapting Function Points to contemporary software systems: A review of proposals", 2nd Australian Conference on Software Metrics. Australian Software Metrics Association, 1995.

[14] C. Jones, Applied Software Measurement - Assuring Productivity and Quality, McGraw-Hill, New York, 1991.

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

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

[17] 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.

[18] S. A. Whitmire, "An Introduction to 3D Function Points", Software Development, Vol. 3 No.4, 1995.

[19] 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.

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

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

[22] S. A. Whitmire, "Applying Function Points to Object Oriented Software Models", in Software Engineering Productivity Handbook, J. Keyes Ed., New York, Windcrest/McGraw-Hill, 1993.

[23] G. Costagliola, F. Ferrucci, G. Tortora, and G. Vitiello, "Class Point: An Approach for the Size Estimation of Object-Oriented Systems", IEEE Transactions On Software Engineering, Vol. 31, No. 1, pp. 52-74, January 2005.

[24] G. Antoniol, C. Lokan, G. Caldiera, and R. Fiutem, "A Function Point-Like Measure for Object-Oriented Software", Empirical Software Engineering, 4 (3), September 1999.

[25] M. Maya, A. Abran, S. Oligny, D. St-Pierre, and J.-M. Desharnais, "Measuring the Functional Size of Real-Time Software" 9th European Software Control and Metrics Conference and 5th Conference for the European Network of Clubs for Reliability and Safety of Software (ESCOM-ENCRESS-98), Rome, Italy, 1998.

[26] ISO/IEC 19761:2003, Software engineering – COSMIC-FFP – A functional size measurement method, Geneva: ISO, 2003.

[27] J.M. Desharnais, P. Morris, "Measuring ALL the Software not just what the Business", IFPUG Conference, 1998.

[28] S. Oligny, J.M. Desharnais, A. Abran, "A Method for Measuring the Functional Size of Embedded Software", 3rd Int. Conf. on Industrial Automation, pp. 7-9, 1999.

[29] L. Lavazza, V. del Bianco, and Geng Liu. "Analytical convertibility of functional size measures: a tool-based approach."" Joint Conference of the 22nd Int. Workshop on Software Measurement and the 7th Int. Conf. on Software Process and Product Measurement. IEEE Computer Society, 2012.

[30] Total Metrics. Glossary of metrics terms. at http://www.totalmetrics.com/resources/software-metrics-glossary (accessed on May 16th, 2014).