A case study using the COSMIC-FFP Measurement Method for Assessing Real-Time System Specifications

Manar Abu Talib[1], Adel Khelifi[2], Alain Abran[3] and Olga Ormandjieva[4]

[1]Zayed University, P.O. Box 4783, Abu Dhabi, UAE
[2]Al Hosn University, P.O. Box: 38772, Abu Dhabi, UAE
[3]École de technologie supérieure, Université du Québec, Montreal, Canada
[4]Concordia University, 1455 de Maisonneuve Blvd. W., Montreal, Canada
manar.talib@zu.ac.ae, a.khelifi@alhosnu.ae, aabran@ele.etsmtl.ca,
ormandj@cse.concordia.ca

Abstract. The success of a system development project largely depends on the nonambiguity of its system-level requirements specification document, where the requirements are described at the system level rather than at the software and hardware level. There may be missing details about the allocation of functions between hardware and software, both for the developers who will have to implement such requirements later on, and for the software measurers who have to immediately attempt to measure the software functional size of such requirements.. The result of different interpretations of the specification problem would lead to different software being built, and of different functional size. The research described in this paper is concerned with the challenges inherent in understanding the initial system requirements in textual form and assessing the codesign decisions using the functional size measurement. This paper aimed at understanding the applicability of the COSMIC-FFP functional size measurement method in assessing the hardware-software requirements allocation, and illustrates the approach on a Steam Boiler Controller case study.

Keywords: COSMIC-FFP, ISO 19761, system-level requirements specification, codesign, functional size measurement.

1 Introduction

Writing system requirements that unambiguously define the hardware/software allocation of the functionality is critical in the system life cycle. If not detected early, ambiguities can lead to misinterpretations at the time of requirements analysis and specification, or at a later phase of the software development life cycle, causing an escalation in the cost of requirements elicitation and software/hardware development. Detecting ambiguities at an early stage of the system requirements elicitation process can therefore save a great deal of aggravation, not to mention cost. The importance of detecting ambiguity earlier in the system development process is also outlined in IEEE Standard 830-1998 [1], which describes the practices, recommended by the IEEE for writing a Software Requirements Specification (SRS) document, and defines

the quality characteristics of a "good" SRS document. These are that the requirements be: (1) Correct, (2) Unambiguous, (3) Complete, (4) Consistent, (5) Ranked according to importance, (6) Verifiable, (7) Modifiable and (8) Traceable. Here, "unambiguous" as defined by the standard means that each of the statements in an SRS document has only one interpretation. The IEEE standard further mentions that the inherently ambiguous nature of natural language can make the text of an SRS document fail to comply with the above definition, making it ambiguous, and thereby degrading the overall quality of the document.

Even though the documented requirements used for many case studies for real-time systems come from known sources, such as universities and trusted industrial organizations, there is no documented information about the quality of these requirements.

Specifically, in the documentation of the available case studies, there is generally no claim that their sets of documented requirements meet some quality criteria, such as those specified in IEEE 830. When the requirements do not meet such quality standards, it means that there may be unclear text or missing details from the specification problem, which would impact:

- the developers, who would have to implement such requirements later on, and
- the measurers, who have to measure the software functional size of such requirements.

The result of different interpretations of the specification problem would lead to different software being built, and of different functional size.

In recent work aimed at measuring the software functional size of real-time requirements case studies of unknown quality, measurers have found it necessary to make some assumptions about the specification problem in order to clarify the software requirements. This was because the specification problems in these case studies had been described at the system level, which meant that what was to be done by the hardware and what was to be done by the software was not clearly spelled out.

The research described in this paper is concerned with the challenges inherent in understanding the initial system requirements in textual form and assessing the codesign decisions using the functional size measurement. Our hypothesis is that functional size measurement feedback will help the developers in their tradeoff analysis when allocating functionality to software and hardware. Our approach is based on the COSMIC-FFP method, which not only helps clarify the allocation process while modeling system functionality, but also provides theoretically valid, and thus objective, size measurement results. Based on the functional size results, the effort associated with a given allocation can be further assessed.

The remainder of this paper is organized as follows: section 2 provides background information on the COSMIC-FFP (ISO 19761) measurement method; section 3 introduces the steam boiler case study; section 4 identifies how requirements at the system level and related assumptions made based on unclear text can lead to different functional sizes of the desired software; and, finally, section 5 presents a discussion and observations.

2 Background

This section introduces key notions of COSMIC-FFP as a functional size measurement method.

The COSMIC-FFP functional size measurement method [3] was developed by the Common Software Measurement International Consortium (COSMIC) and is a recognized international standard (ISO 19761 [2]). Its was developed to address some of the major weaknesses of earlier methods, like FPA [4], for example, the design of which dates back almost 30 years to a time when software was much less varied.

In the measurement of software functional size using COSMIC-FFP, the software functional processes and their triggering events must be identified. The unit of measurement in this method is the data movement, which is a base functional component that moves one or more data attributes belonging to a single data group. Data movements can be of four types: Entry (E), Exit (X), Read (R) or Write (W). The functional process is an elementary component of a set of user requirements triggered by one or more triggering events, either directly or indirectly, via an actor. The triggering event is an event occurring outside the boundary of the measured software and initiates one or more functional processes. The subprocesses of each functional process constitute sequences of events, and a functional process comprises at least two data movement types: an Entry plus at least either an Exit or a Write. An Entry moves a data group, which is a set of data attributes, from a user across the boundary into the functional process, while an Exit moves a data group from a functional process across the boundary to the user requiring it. A Write moves a data group lying inside the functional process to persistent storage, and a Read moves a data group from persistent storage to the functional process. See Figure 1 for an illustration of the generic flow of data groups through software from a functional perspective.

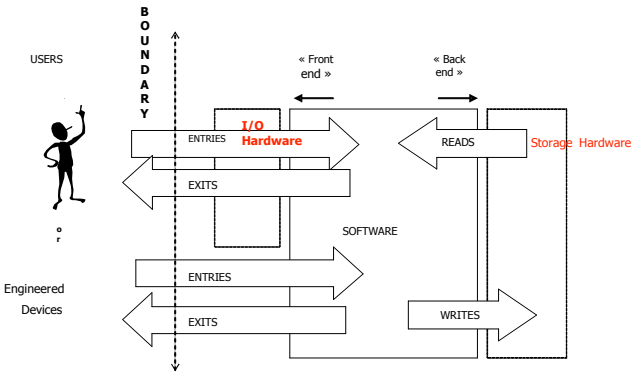

Fig. 1. Generic flow of data through software from a functional perspective [2].

3 Case Study: Steam Boiler

The Steam Boiler Control specification problem of J. R. Abrial and E. Brger [5] was derived from an original text by J. C. Bauer for the Institute for Risk Research at the University of Waterloo, Ontario, Canada. The original text had been submitted as a competition problem to be solved by the participants at the International Software Safety Symposium organized by the Institute for Risk Research. It provides the specification design that will ensure safe operation of a steam boiler by maintaining the ratio of the water level in the boiler to the amount of steam emanating from it with the help of the corresponding measurement devices. The Steam Boiler System consists of the following physical units:

- Steam Boiler: the container holding the water;
- Pump: the device for pouring water into the steam boiler;
- Valve: the mechanism for evacuating water from the steam boiler;
- Water Level Measurement device: a sensor to measure the quantity of water q (in liters) and inform the system whenever there is a risk of exceeding the minimum or maximum amounts allowed.

Figure 2 shows the Steam Boiler and the relationships between its components. The Steam Boiler is assumed to start up with a safe amount of water. The Controller runs a control cycle every 5 seconds to check on the amount of water currently in the system, and then triggers the Water Level Measurement device and sends the result to the Controller. The Controller receives the current level and checks whether it is normal, above normal or below normal: if the water level is normal, it will do nothing; if there is a risk that the minimum safe level will be reached, the Pump will be triggered to pour more water into the Steam Boiler; and if there is a risk that a level higher than normal will be reached, the Valve will be triggered to evacuate water from the Steam Boiler.

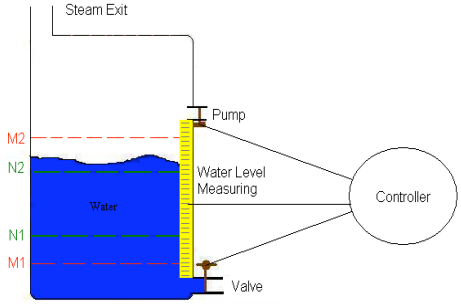

Fig. 2. Steam Boiler controller.

4 Identification of Software-Related Ambiguities in the Specifications

The way the specification problem is written is ambiguous from a software viewpoint: at the system level, the specification text talks about a single controller, which is the

system controller; however, in practice, this system controller consists of two controllers: a hardware part and a software part, which are not specified at the system level. The specifications about the interactions between the hardware and the software are sometimes ill-defined in real-time applications: what is really to be done by the hardware, and what is really to be done by software? For instance, in this case study, the above requirements are at the system level, not at the software level. For this case study, a number of hardware/software allocation alternatives of can be proposed, with their corresponding specific requirements (see Table 1).

Table 1. Hardware/software allocation alternatives.

#.	Hardware controller	Software controller
1	Generates the five-second signal Activates the measuring device Reads the output of the measuring device Determines if it is at min, max or normal Sends the value (min, max or normal) to the software controller	Receives the current water level signal value (min, max or normal) Based on the values received, it activates the pump or valve.
2	Generates the five-second signal Activates the measuring device Reads the output of the measuring device Sends the reading to the software controller	Receives the current water level signal value Determines if it is min, max or normal Based on analysis (min, max or normal), it activates the pump or valve. Comment: From the system requirements, it is not clear whether this min-max is constant or variable based on some context. If it is constant, this should be clarified in the system's software requirements. If it varies, then additional software requirements are needed to provide the ability to manage/update it (see the 4th option)
3	Receives the get-level signal to activate the measuring device Activates the measuring device Reads the output of the measuring device Sends the reading to the software controller	Generates the five-second signal Generates the get-level signal Receives the current water level signal value Determines if it is min, max or normal Based on analysis (min, max or normal), it activates the pump or valve.
4	Additional options could be generated: for example, a database containing the min-max values, which can be updated by human users.	

From the software viewpoint, the text about the controller in the specification problem is ambiguous: for instance, how will the software controller determine whether it is a min or a max?

Alternative 1- The min-max values are specified as constants: then an Entry of these values is needed for the software controller.

Alternative 2- The min-max values are specified as stored values: then a Read of these stored values is needed for the software controller.

Alternative 3- The min-max values are specified as stored updatable values: then an update function is needed (with corresponding data movements).

Alternative 4 - Additional options could be specified: for example, there could be a requirement for a database containing the min-max values, which can be updated by a human operator.

Alternative 1 states that it is the hardware part of the controller that reads the water level, makes the decision (calculation) about the min-max (and the risk of getting close to the min-max) and then sends the outcome (min, max or normal) to the software part of the controller. The software controller is then only responsible for sending close/open messages to the valve and pump.

This alternative thus describes the interactions of the software controller with other components, that is, when the water level is below the minimum, is normal or is above the maximum. This interaction begins when it receives a signal from the hardware measurement device (under the control of the hardware controller every 5 seconds). The basic flow therefore is described as follows (see Figure 3):

1. The software controller receives data from the hardware controller.
2. The software controller obtains the water level measurement outcome (min, max or normal).
3. The software controller sends an open/close message to the pump, which reacts accordingly.
4. The software controller sends a close/open message to the valve.

The sequence diagram for this alternative is presented in Figure 3.

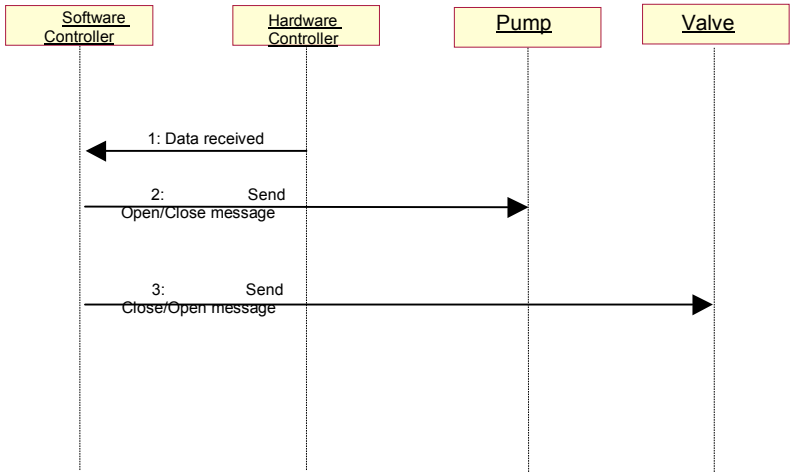

Fig. 3. Alternative 1 – Sequence diagram of interactions of the software controller with other components.

This alternative 1 would lead to a functional size of 3 Cfsu (Cfsu = COSMIC functional size units) for the corresponding software (see Table 2).

Table 2. List of COSMIC-FFP data movements – Alternative 1.

Process description	Triggering event	Sub-process Description	Data Group	Data movement Type	Cfsu
Maintain water level	Water level signal	Obtain water level measurement (value = below normal, normal or above normal)	Controller Sensor	E	1
		(Logic) Check if any action is needed; if not, terminate the cycle	Controller		
		Send message to pump (value = open or close)	Controller Pump	X	1
		Send message to valve (value = open or close)	Controller Valve	X	1
Total functional size in Cfsu					3 Cfsu

Alternative 3, as the next example, states that it is the hardware part of the controller that receives the five-second signal from the software controller to activate the water level measuring device. Then, the hardware reads the water level and makes the decision (calculation) about the min-max (and the risk of getting close to it), and then sends the outcome (min, max or normal) to the software part of the controller. The software controller is then responsible for generating the five-second signal, activating the measuring device and sending close/open messages to the valve and pump.

The interaction starts when it receives data from the water level measurement device (under the control of the software controller every 5 seconds). Figure 4 shows the basic flow of such an interaction, as follows:

1. The software timer sends the 5-second signal to software controller.
2. The software controller sends a get-level request for the current water level to the hardware controller.
3. The software controller obtains the current water level from the hardware controller.
4. The software controller reads the range of the water (min to max) and compares the current water level with the min and max.
5. The software controller checks if any action is needed; if not, the cycle is terminated.
6. The software controller sends the new status to the pump (value = open or close).
7. The software controller sends the new status to the valve (value = open or close).

This alternative 3 would lead to a functional size of 6 Cfsu for the corresponding software – see Table 3.

Similarly, other alternative mixes of hardware/software functions for the same specification problem would lead to different software being built, each of a different functional size.

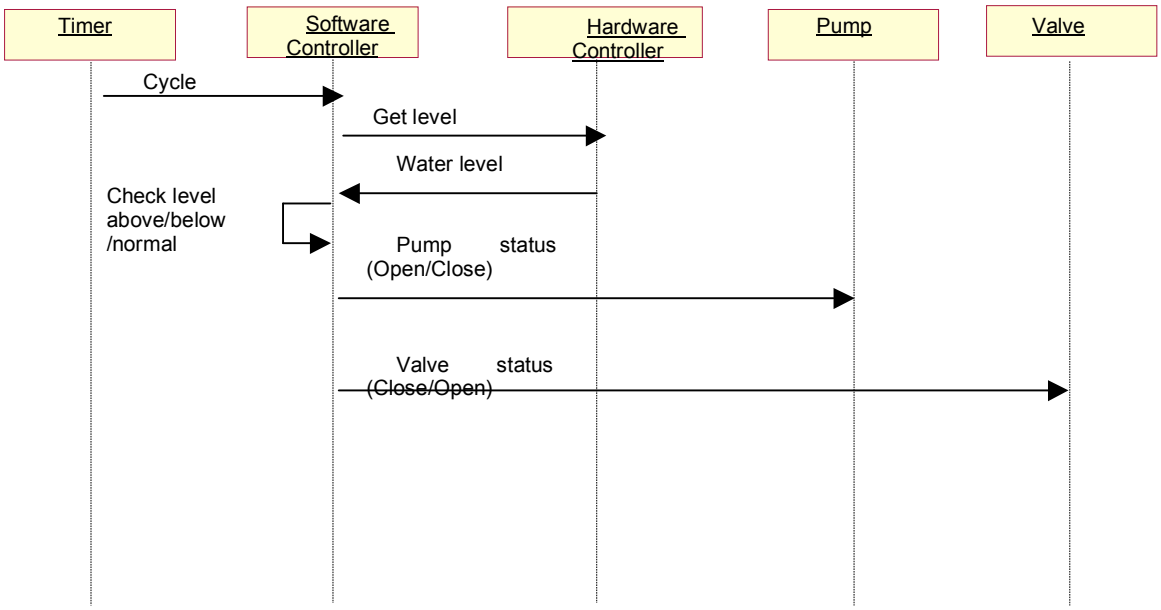

Fig. 4. Alternative 3 – Sequence diagram of interactions of the software controller with other components.

Table 3. List of COSMIC-FFP data movements – Alternative 3.

Process description	Triggering event	Data Movement Description	Data Group	Data movement Type	Cfsu
Maintain water level	5-second signal	Send 5-second signal to Controller	5-second signal	E	1
		Request current water level	Get level signal	X	1
		Obtain current water level	Water level signal	E	1
		Read the range of the water (min to max) and compare the current water level with the min and max.	Water level range	R	1
		(Logic) Check if any action is needed; if not, terminate the cycle			
		Send new status to pump (value = open or close)	Pump status signal	X	1

		Send new status to valve (value = open or close)	Valve status signal	X	1
Total functional size in Cfsu					6 Cfsu

5 Discussion

When a requirements case study is written at the system level, there may be missing details about the allocation of functions between hardware and software, both for the developers who will have to implement such requirements later on, and for the software measurers who have to immediately attempt to measure the software functional size of such requirements. As a result, different interpretations of a specification problem would lead to different functional sizes for the corresponding software.

This paper has explored in the well known steam boiler case study different alternatives for hardware/software allocation from the system requirements. Implementing alternative 1 in the steam boiler application, for example, the total functional size is 3 Cfsu, with a different allocation of functions between hardware and software, as stated in alternative 3, resulting in a total functional size of 6 Cfsu. Therefore, the choice among the alternatives will affect what software will be built and, correspondingly, what its final functional size will be. It becomes clear, then, that different interpretations can be derived from the same steam boiler specification problem because of missing details regarding the software and hardware requirements.

These findings are significant from three perspectives:

- The writers of such case studies should clearly spell out that their case studies are documented at the system level, that the hardware/software function allocation has not been specified and that the quality of these requirements is not documented and should not be assumed to meet the IEEE-830 quality criteria.
- The users of such case studies should be cautious when they use them for studies related to software functions; the findings observed from the use of a case study will depend on the specific interpretation of its users, and it might not be possible to generalize them to all potential interpretations of that specific case study, as doing so might lead to confusion in the minds of readers, all the more so if the assumptions about the hardware/software allocation of functions has not been documented. The users of such a case study should therefore verify whether it has been documented at the system level or at a level documenting the hardware/software allocation of functions. Users should also be aware that, unless it is been specifically documented at the function allocation level, such a case study will not necessarily meet the IEEE-830 quality criteria.
- The measurers of such case studies should also be cautious when they use them for studies related to the measurement of software functions, and for the same reasons that the users of these case studies should exercise caution: different allocation of hardware/software functions can lead to different software functional sizes, and, unless the assumptions and interpretations are clearly

spelled out and documented during the measurement process, it is difficult to justify a specific measurement result and to demonstrate that it is the correct size for the specific mix of hardware/software functions. Indeed, measurement results without detailed documentation of the assumptions on which the measurement is based could lead, perhaps wrongly, to a lack of confidence in the measurement results themselves, in the measurers ability to come up with correct and repeatable measurement results, and, ultimately, in the potential for a measurement method to lead to repeatable results.

A key insight from this study is that the measurement of the software size of a specification problem can make a number of positive contributions if the measurer clearly documents both the ambiguity he has found in a specification document and the assumptions he has made, such as for the hardware/software allocation of functions, when measuring the functional size of the software.

In our view, it should be the responsibility of the measurer to identify, within the documented requirements, what he considers as ambiguities and omissions, and to document the assumptions he made that led to the measurement results he documented. These comments, documented by the measurer, represent a value-added contribution to quality during the measurement process and possibly help reduce costs later on in the development project. Of course, the measurer's observations and comments should subsequently be reviewed by the project manager, who should then address these comments prior to proceeding further with the project.

Further work is in progress to analyze additional case studies, verify these observations and derive techniques that would improve both the measurement process and the contributions to the improvement of the quality of the problem specifications.

Acknowledgements

This research project has been funded partially by the European Community's Sixth Framework Programme – Marie Curie International Incoming Fellowship under contract MIF1-CT-2006-039212.

References

1. IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements SpeciPcations, Software Engineering Standards Committee of the IEEE Computer Society (1998)
2. Abran, A., Desharnais, J.-M., Oligny, S., St-Pierre, D. and Symons, C.: COS-MIC FFP – Measurement Manual (COSMIC implementation guide to ISO/IEC 19761:2003). École de technologie supérieure – Université du Québec, Montréal (2003)
3. ISO/IEC 19761. Software Engineering – COSMIC-FFP – A functional size measurement method. International Organization for Standardization – ISO, Geneva (2003)
4. Albrecht, A.J. and Gaffney, J.E.: Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. IEEE Trans. Software Eng. vol. SE-9, no. 6, pp. 639-648 (1983)
5. J. R. Abrial: Steam Boiler Control Specification Problem (1994)