2010 Eighth ACIS International Conference on Software Engineering Research, Management and Applications

A Generic Model for the Specification of Software Interface Requirements and Measurement of their Functional Size

Khalid T. Al-Sarayreh
Software Engineering Department, (ETS)
University of Quebec
1100 Notre-Dame West, Montréal, Québec H3W 1T8
Canada
khalid.al-sarayreh.1@ens.etsmtl.ca

Alain Abran
Software Engineering Department, (ETS)
University of Quebec
1100 Notre-Dame West, Montréal, Québec H3W 1T8
Canada
alain.abran@etsmtl.ca

***Abstract*—The European ECSS-E-40 series of standards for the aerospace industry includes interfaces as one of 16 types of non functional requirement (NFR) for embedded and real-time software. An interface is typically described at the system level as a non functional requirement, and a number of concepts and terms are provided in that series to describe various types of candidate interfaces. This paper collects and organizes these interface-related descriptions into a generic model for the specification of software interface requirements, and to measure their functional size for estimation purposes using the COSMIC ISO 19761 standard.**

Keywords-Interface requirements; Non functional requirements (NFR); Functional size; COSMIC – ISO 19761; ECSS International Standards; Software interface measurement.

I. INTRODUCTION

Non functional requirements (NFR) are vital to the quality and success of software systems. Identification and specification of NFRs early in the software life cycle are therefore of prime importance in defining a comprehensive specification, and subsequent evaluation, of software product quality. Similarly, these software-related NFRs should be taken into account when estimating software projects.

In practice, NFR may be viewed, defined, interpreted, and evaluated differently by different people, particularly when they are stated briefly and vaguely [1], [2]. NFRs can also be relative, since their interpretation and importance may vary, depending on the particular system being considered [2]. In addition, NFRs can often interact with one another: attempts to achieve one NFR, for example, can hurt or help the achievement of others [3].

To achieve the system NFR described typically at a fairly high level, the system engineers must next specify what has to be allocated at either the hardware or at the software level.

While a number of researchers are investigating the field of NFR, standards organizations have attempted over the years to identify lower levels of information in terms of requirements that must be implemented through hardware and software. In related works, the requirements are initially addressed typically at the system level [4], [5], [6] and [7] as either high-level system functional user requirement (system-FUR) or high level system non-functional requirements (system-NFR); such high level requirements must typically next be detailed and allocated to specific-related functions which may be implemented in either or both hardware and software, as software FUR (Soft-FUR) for instance – see Figure 1.

For example, a system-FUR will describe what are the required functions needed in a system, while a system-NFR will describe how the required functions must behave in a system [8], [9], [10], [11], and [12]; in the software requirements engineering step, such system-NFR may next be detailed and specified as software-FUR to allow a software engineer to develop, test and configure the final deliverables to system users.

"Functional" refers to the set of functions the system is to offer, while "non-functional" refers to the manner in which such functions are performed. Functional user requirements (FUR) are typically phrased with subject or predicate constructions, or noun/verb, such as: "The system must print 5 reports". Non-functional requirements (NFR) are typically phrased with adverbs or modifying clauses, such as: "The system will print 5 reports quickly" or "The system with print 5 reports with high reliability".

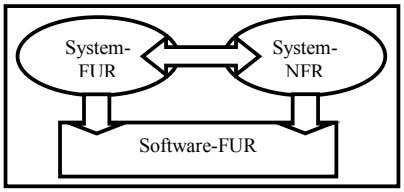

Figure 1. Mapping system into software-FUR

Currently, there exists no generic model for the identification and specification of software interface requirements from the various views documented in international standards and in the literature. Consequently, it is challenging as well to measure them and to take them into account quantitatively for estimation purposes.

This paper reports on the work carried out to define an integrated view of software interface requirements on the basis of international standards, and on the use of a generic model of interface software-FUR to measure their functional size using the COSMIC measurement standard, independently of software development and implementation methodologies and technologies.

This paper is organized as follows. Section 2 presents a generic view of software-FUR in ISO. Section 3 identifies the standards describing interfaces

978-0-7695-4075-7/10 $26.00 © 2010 IEEE
DOI 10.1109/SERA.2010.35

IEEE computer society

Authorized licensed use limited to: ECOLE DE TECHNOLOGIE SUPERIEURE. Downloaded on June 21,2010 at 17:05:41 UTC from IEEE Xplore. Restrictions apply.

requirements. Section 4 presents a standard-based definition of a generic model of a software interfaces requirements. Section 5 presents the sizing of a reference instantiation of the generic model an interface-software-FUR. Finally, a discussion is presented in section 6.

II. A GENERIC VIEW OF SOFTWARE-FUR IN ISO

In the collection of ISO standards, it is specified in the ISO 14143-1 [13] that a functional size measurement method must measure the software-FUR. In addition, ISO 19761 – COSMIC [14] proposes a generic model of software-FUR that clarifies the boundary between hardware and software. Figure 2 illustrates the generic flow of data from a functional perspective from hardware to software. From this generic model of software requirements in Figure 2 the followings can be observed:

- Software is bounded by hardware. In the so-called "front-end" direction (i.e. left-hand side in Figure 2) software used by a human user is bounded by I/O hardware such as a mouse, a keyboard, a printer or a display, or by engineered devices such as sensors or relays. In the so-called "back-end" direction (i.e. right-hand side of Figure 2), software is bounded by persistent storage hardware like a hard disk and RAM and ROM memory.
- The functional flow of data groups can be characterized by four distinct types of movement. In the "front end" direction, two types of movement (ENTRIES and EXITS) allow the exchange of data with the users across a 'boundary'. In the "back end" direction, two types of movement (READS and WRITES) allow the exchange of data with the persistent storage hardware.
- Different abstractions are typically used for different measurement purposes. In real-time software, the users are typically the engineered devices that interact directly with the software that is, the users are the 'I/O hardware'. For business application software, the abstraction commonly assumes that the users are one or more humans who interact directly with the business application software across the boundary; the 'I/O hardware' is ignored.

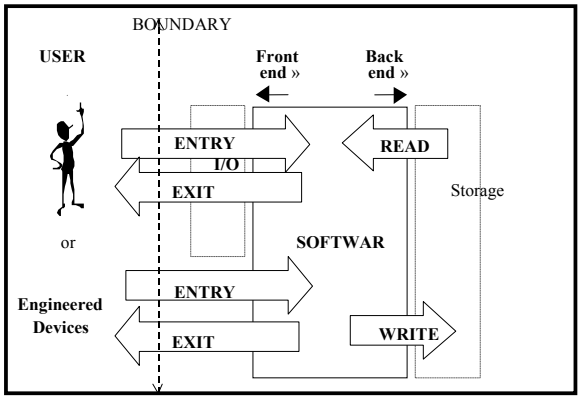

Figure 2. Generic flow of data groups through software from a functional perspective in COSMIC – ISO 19761

As an FSM method, COSMIC is aimed at measuring the size of software based on identifiable Functional User Requirements. Once identified, those requirements are allocated to hardware and software from the unifying perspective of a system integrating these two "components". Since COSMIC is aimed at sizing software, only those requirements allocated to the software are considered.

III. IDENTIFICATION OF STANDARDS DESCRIBING INTERFACES REQUIREMENTS

This section presents a survey of the interface-related views, concepts and terms in the ECSS and IEEE-830 standards. This section identifies which standards currently address some aspects of the software-FUR derived from system-NFR, specifically for the Interfaces software-FUR – see Figure 3. The expected outcome is the identifications of the various elements that should be included in the design of a standard-based framework for modelling software FUR for interfaces.

The ECSS standards of the European Space Agency [15], [16], and [17] present the software interfaces as a set of NFRs for real-time and embedded software. In particular, the ECSS-E-40 series [18] includes the design of the external interface as part of the interface control document (ICD), while the design of the internal interface is included as part of the software design document (SDD); [18] also specifies that the detailed design of the software product interfaces should be defined during the interface design phase.

In these standards, the majority of interfaces are software-to-software interfaces, and the ECSS requires that they shall be defined in the requirements baseline (i.e. the requirements base must include the requirements applicable to the various elements of the system product tree [15]). The elements of interfaces are dispersed in various system views throughout different ECSS standards and are expressed as either:

- System interface functional user requirements (interface system-FUR)
- System interface non-functional requirements (interface system-NFR)

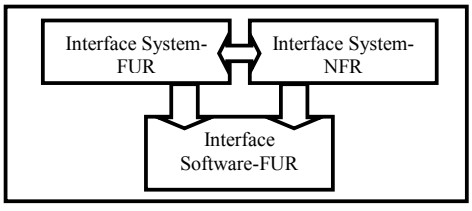

Figure 3. Mapping system into software-FUR for interfaces

In the ECSS standards the following set of concepts and vocabulary are used to describe interfaces:

- Application software,
- Hardware configuration,
- Interface communication,
- Interface specifications through programming languages,

Authorized licensed use limited to: ECOLE DE TECHNOLOGIE SUPERIEURE. Downloaded on June 21,2010 at 17:05:41 UTC from IEEE Xplore. Restrictions apply.

- Database management system,
- Hardware interfaces.

While conducting this inventory of interface concepts and terms described in the ECSS-E-40 and ECSS-Q-80 series and in ECSS-ESA as the integrated standard for ECSS-E and ECSS-Q, it was observed that:

- The interface elements are dispersed throughout various parts, and there is therefore no integrated view of all types of candidate interfaces;
- These various interface elements are described differently, and at different levels of detail;
- There is no obvious link in ECSS-ESA between the interface control documents (ICD), where the external interfaces are defined, and the interfaces in the software design document (SSD), where the internal interfaces are defined.
- There is no obvious guidance on how to measure the interface as NFR.

Software interface requirements are also presented in IEEE 830 [19] as NFR, and the interface is defined through a detailed description of all inputs into, and outputs from, the software system. In particular, IEEE-830 [19] mentions that the interface can be analyzed and understood through the user interfaces, the hardware interfaces, the software interfaces, and the communications interfaces. It is to be noted that when IEEE-830 defines an interface as 'inputs' or outputs', these are considered only as collections of data movements and not integrated into specific functional processes. IEEE 830 does not provide an analysis or explanations linking the identified set of interface concepts

It is also to be noted that neither ECSS nor IEEE-830 proposes a way to measure such software interfaces requirements, and that, without measurement, it is challenging to take such NFRs as quantitative inputs to an estimation process or in productivity benchmarking.

IV. A STANDARD-BASED DEFINITION OF A GENERIC MODEL OF SOFTWARE INTERFACES REQUIREMENTS

This section identifies first, and assembles next, the dispersed terminologies and concepts of interfaces dispersed throughout ECSS, IEEE and ISO standards into a proposed model of interface software-FUR – see Figure 3, through the use of the generic model of FUR proposed in the COSMIC model. This COSMIC-based generic model can then become a framework for describing the software interface requirements (i.e. from system-NFR into software-FUR) based on ECSS.

A. Definition of a software interface

The following definition has been adopted in this research:

- Software interface: a layer of utility software that sits between application software and systems to transparently integrate distinct technologies. According to (ECSS-Q-40-04A Part 1) an internal interface could also include hardware and system requirements.

1) Types of interface requirements

Four types of interface requirements can be identified:

- Functional user interface requirements: what is needed to allow users (devices and humans) to interact with the system;
- Communication interface requirements: what is needed to allow communication with other systems or components between two pieces of software;
- Software interface requirements: what is needed to allow communication with other software system components that are not within the software to be designed (such as operating system, files, database management system, or other applications software);
- Hardware interface requirements: what is needed to ensure support for the hardware and the specific hardware configuration by the system (i.e. logical structure, physical address, and expected behaviour)

2) Interface functionality to be specified (entities)

The interface functionality to be specified (and corresponding entities to be measured [20]) is divided into external and internal functionality - see Table I. At times, this interface functionality may be within different layers of software requirements.

TABLE I: Software Interface Functionality (Entities)

Interface Functionality Type	Interface Functionality
External Interface functionality	• Software application (GUI for example) • Hardware configuration • Communication links
Internal Interface functionality	• Interface specifications • Software modules • Socket (API for example)

B. Identification of the relationships across the software interface functionality

This sub-section identifies the component types and functional relationships in the software interfaces.

1) Identification of the interface functionality types in software

- **Interface Functional Type 1: Hardware Configuration and Software Modules**

Each functional process in a hardware configuration can interact with at least one, and perhaps more, software modules - see Figure 4.

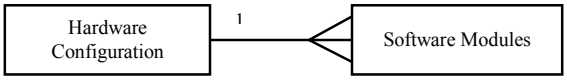

Figure 4. Hardware configuration and software modules

Authorized licensed use limited to: ECOLE DE TECHNOLOGIE SUPERIEURE. Downloaded on June 21,2010 at 17:05:41 UTC from IEEE Xplore. Restrictions apply.

- **Interface Functional Type 2: Software Applications and Software Modules**

Each functional process in a software application interacts with at least one, and perhaps more, software modules - see Figure 5.

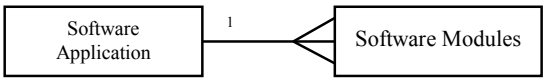

Figure 5. Software application and software modules

- **Interface Functional Type 3: Communication Links and Software Modules**

Each functional process in a communication link interacts with at least one, and perhaps more, software modules - see Figure 6.

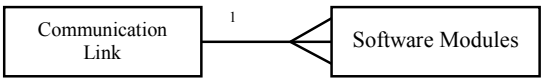

Figure 6. Communication link and software modules

- **Interface Functional Type 4:**

The internal interface is restricted to the software-to-software interface between two software components within the same software item - see Figure 7.

– Each functional process of an interface specification link can input an entry, and output an exit from data groups (DG1, DG2…DGn).
– An Interface specification link receives data from a set of data socket groups, then reads and writes these data using an interface method.
– An Interface specification link sends or exits these data to a software module.

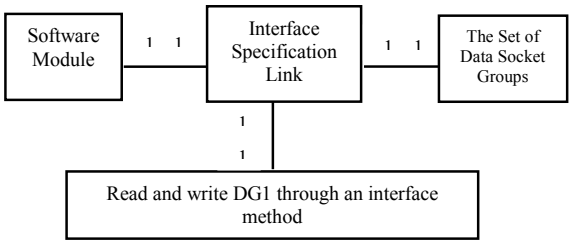

Figure 7. Software-software interface

2) Consolidation of the functional relationships in the software interfaces

Figure 8 presents a consolidated view of the relationships between the functional types in the software interfaces, using the COSMIC model for graphical representation. More specifically:

- The sub-model of Interface Functional type 1 can be used to specify (and to measure the functional size of) the external interface between the hardware configuration and the internal interface (represented inside the boundary in Figure 8).
- The sub-model of Interface Functional type 2 can be used to specify (and to measure the functional size of) the external interface between the software application and the internal interface.
- The sub-model of Interface Functional type 3 can be used to specify (and to measure the functional size of) the external interface between the communication links and the internal interface.
- The sub-model of Interface Functional type 4 can be used to specify (and to measure the functional size of) the internal interface between the specification link and the software module(s) and between the set of data sockets data groups of the interface (inside the boundary in Figure 8).

This model is referred here as a generic model of interface software-FUR.

TABLE II: Interface Functional Size Measurement with respect to the Generic Reference Model of Interface-Software-FUR

Interface Functional type	Data Movement Description	Data Movement Type
Hardware Configuration	• A software module RECEIVES a data group hardware configuration	**E**
	• A software module SENDS a data group to a hardware configuration	**X**
Software Application	• Software module RECEIVES a data group from a software application	**E**
	• Software module SENDS a data group to a software application	**X**
Communicatio -n Links	• A software module RECEIVES a data group from a communication link.	**E**
	• A software module SENDS a data group to a communication link	**X**
Software Module	• A software module RECEIVES a data group from the interface specification link.	**E**
	• A software module READS a data group from an interface component	**R**
	• A software module WRITES a data group to an interface component (Functions, Modules, and Code)	**W**
	• Either a software module SENDS a data group to an interface specification (inside the boundary)	**X**
	• A software module SENDS a data group to a hardware configuration	**X**
	• A software module SENDS a data group to a software application	**X**
	• A software module SENDS a data group to a communication link.	**X**
Interface Specification Link (Socket Interface Programming)	• An interface specification link RECEIVES a data group from the data socket	**E**
	• An interface specification link READS a data group from the data socket	**R**
	• An interface specification link WRITES a data group using a method on a port	**W**
	• An interface specification link SENDS a data group to software programming	**X**
Data Groups (Data Socket)	• A data socket RECEIVES a data Group from the interface specification	**E**
	• A data socket SENDS a data group to the interface specification	**X**
The Total Size in COSMIC		**19 CFP**

Authorized licensed use limited to: ECOLE DE TECHNOLOGIE SUPERIEURE. Downloaded on June 21,2010 at 17:05:41 UTC from IEEE Xplore. Restrictions apply.

V. SIZING A REFERENCE INSTANTIATION OF THE GENERIC MODEL OF AN INTERFACE-SOFTWARE-FUR

The specification of interface-software-FUR in any specific project is a specific instantiation of the proposed generic model of interface-software-FUR as described in Figure 8. When the specification document is at the level of the movements of data groups, then these interface requirements can be directly measured using the COSMIC measurement rules.

Table II presents next the measurement results using COSMIC of a specific instantiation of interface requirements which would have one of each of the entities and relationships described in section IV and Figure 8 (note: for measurement purposes, the 'interface functional type represents the entity to be measured).

For example, for a hardware configuration (functional type 1):

- A software module RECEIVES a data group from a hardware configuration.
- A software module SENDS a data group to a hardware configuration.

This requirement corresponds to a COSMIC Entry and to a COSMIC Exit data movement, for a functional size of two COSMIC Function Points, or 2 CFP. The corresponding total functional size of this specific instantiation would therefore consider 19 data movements of one data group, which would then gives a functional size of 19 CFP with the COSMIC ISO 19761 standard - see Table II, bottom line.

VI. DISCUSSION

This paper has introduced a procedure for specifying and measuring the requirements of the software for the internal and external interfaces needed to address the system's non functional requirements for interfaces.

The main contribution of this paper is our proposed Generic Model of Interface software functional user requirements. This generic model is considered as a kind of reference model for specifying interface-Software-FUR from system Interface-NFR, as well as for measuring the functional size of software interfaces. It is based on:

- The ECSS and IEEE 830 standards for the description of interfaces requirements; and
- The COSMIC generic model of software FURS

The model is independent of the software type and the programming languages in which these interfaces will be implemented.

The proposed generic interface model (i.e. reference model) provides:

- A specification model for each type, or all types, of interface requirements: for example, software interfaces for application programming, communication links, and hardware configuration.
- A specification measurement model for each type, or all types, of interface requirements.

Future work includes verification and evaluation of this generic model to ensure full coverage of interface requirements, and verification with group of experts to develop a consensual generic model which could be proposed as a candidate for standardization.

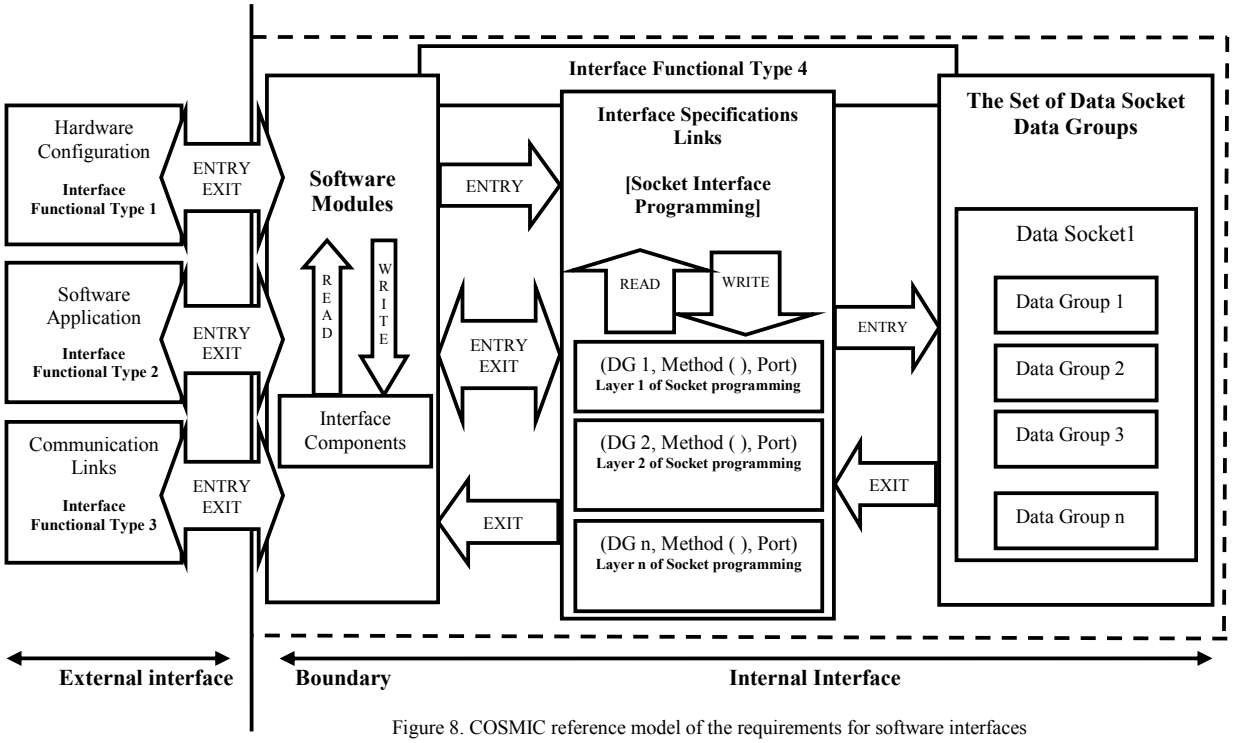

Figure 8. COSMIC reference model of the requirements for software interfaces

Authorized licensed use limited to: ECOLE DE TECHNOLOGIE SUPERIEURE. Downloaded on June 21,2010 at 17:05:41 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] L. Chung and J. C. S. d. P. Leite, "On Non-Functional Requirements in Software Engineering", in "Conceptual Modeling: Foundation and Applications, Essays in Honor of John Mylopoulos", 2009.

[2] L. Chung, B.Nixon, E.Yu, J. Mylopoulos, "Non-Functional Requirements in Software Engineering", Springer, Heidelberg, 1999.

[3] J. Mylopoulos, L.Chung, B.Nixon, "Representing and Using Nonfunctional Requirements: A Process-Oriented Approach", IEEE Transactions on Software Engineering vol. 18, pp. 483-497, 1992.

[4] A. I. Antón, "Goal identification and refinement in the specification of software-based information systems", PhD Thesis, Georgia Institute of Technology, 1997.

[5] A. M. Davis, "Software requirements: objects, functions, and states", Prentice-Hall, Inc., 1993.

[6] I. Jacobson, G. Booth, J. Rumbaugh, "Excerpt from the Unified Software Development Process: The Unified Process", IEEE Software, vol. 16, pp. 96-102, 1999.

[7] K. Wiegers, "Software Requirements", 2nd edition. Microsoft Press, 2003.

[8] G. Roman, "A Taxonomy of Current Issues in Requirements Engineering", IEEE Computer, pp. 14-21, 1985.

[9] J. Mostow, "A Problem Solver for Making Advice Operational", National Conference on Artificial Intelligence (AAAI-83), AAAI, Menlo Park, Calif, pp. 279-283, 1983.

[10] B. W. Boehm, "Characteristics of software quality", Amsterdam, New York: North-Holland Pub. Co., American Elsevier, 1978.

[11] R. Thayer, " Software System Engineering: Tutorial", IEEE Computer, pp. 68-73, 2.

[12] M. Shaw, "Larger Scale Systems Require Higher-Level Abstractions", Software Specification and Design, IEEE Computer Society, vol. 14 pp. 143-146, 1989.

[13] ISO/IEC-14143-1, "Information technology - Software measurement - Functional size measurement Part 1: Definition of concepts", International Organization for Standardization, Geneva (Switzerland), 1998.

[14] ISO/IEC-19761, "Software Engineering - COSMIC v 3.0 - A Functional Size Measurement Method", International Organization for Standardization, Geneva (Switzerland), 2003.

[15] ECSS-E-40-Part-1B, "Space Engineering: Software - Part 1 Principles and Requirements", European Cooperation for Space Standardization, The Netherlands 2003.

[16] ECSS-E-40-Part-2B, "Space Engineeing:Software- part 2 Document Requirements Definitions", European Cooperation for Space Standardization, The Netherlands, 2005.

[17] ECSS-Q-80B, "Space product assurance: Software product assurance", European Cooperation for Space Standardization, The Netherlands, 2003.

[18] ESA, "Tailoring of ECSS Software Engineering Standards for Ground Segments in ESA Part C: Document Templates", European space agency / agence spatiale européenne, Paris Cedex (France), June 2005.

[19] IEEE-Std-830, "IEEE Recommended Practice for Software Requirements Specifications", 1993.

[20] A. Abran, "Software Metrics and Software Metrology", Wiley Interscience & IEEE CS Press, to be published in 2010.

Authorized licensed use limited to: ECOLE DE TECHNOLOGIE SUPERIEURE. Downloaded on June 21,2010 at 17:05:41 UTC from IEEE Xplore. Restrictions apply.