An Integrated Measure for Functional Requirements Correctness

Nihal Kececi & Alain Abran
Department of Computer Science
Software Engineering Management Research Laboratory
Université du Québec à Montréal/École de Technologie Supérieure
www.lrgl.uqam.ca

Abstract

This paper describes and illustrates a methodology for identifying the correctness of software functional requirements on the basis of a logic-based dynamic framework. It focuses on the issues related to user and/or system functional requirements; quality attributes, measures and analysis methods, and integrates the core concepts of the Graphical Requirement Analysis (GRA) and COSMIC-FFP techniques:The proposed approach provides a structured procedure for arranging functional software requirements into a graphical framework, thereby providing a means for evaluating their clarity and their presence/absence. Moreover, the architecture of this approach makes it possible to trace specific entities forwards, from system/user requirements to design, and backwards. The way in which the proposed ***I****ntegrated* ***M****easure for* ***F****unctional* ***R****equirements (IMFR) captures critical aspects of functional requirements such as ambiguous or incomplete requirements, incomplete linkages from software requirements to system requirements and to design and/or to test cases is illustrated. Using a sub-system of the Generic Westinghouse Reactor Protection (GWRP) control system case study as an example, we identify and demonstrate various ambiguities of textual software requirements.*

Key words: Requirements, Testing, Measure, Completeness, and Software Quality

1. Introduction

The measurement of the quality of software requirements specifications supports both requirements engineering (definition and analysis) and requirements management (change management, impact analysis, cost estimation, maintainability). This measurement process is, therefore, important for harvesting the benefits associated with the early detection and correction of problems associated with requirements.

A Software Requirement Specification (SRS) has been defined in IEEE Std 730.1-1989 [1] as follows: *"A Software Requirement Specification (SRS) shall clearly and precisely describe each of the essential requirements (functions, performances, design constraints, and attributes) of the software and external interfaces. Each requirement shall be defined such that its achievement is capable of being objectively verified and validated by a prescribed method: for example, inspection, analysis demonstration or test."* Basically. The SRS captures functional and non-functional requirements as well as technical requirements. One or more representatives of the supplier, one or more representatives of the customer, or both, may write (generally using natural language) the SRS.

In recent years, researchers have proposed numerous approaches for specifying/defining, constructing and certifying requirement correctness for high-quality software systems. These proposals include formal methods, semi-formal methods, reviews and analyses, and traceability analysis. Formalizing the requirements (in total or in part) cannot be guaranteed to detect all errors, nor can it ensure that the requirement specifications are correct. Formal review and inspection methods based on checklists and predefined criteria are currently being used as a quality assurance process for requirement specifications. Although there are some advantages to guidelines and checklists, they also have some limitations. For instance, where guidelines are expressed in general terms, their interpretation may depend on expert opinion when objective evaluation is simply not possible. Furthermore, while a checklist can be used to identify potential problems, it cannot be turned into measures; similarly, there is no way to accurately weight the importance of the various recommendations. Checklists are most often used in the absence of available methodologies to address the related issues.

Furthermore, there are two other difficult problems, which are not being addressed either by formal methods or by checklists, and addressing these is the objective of this study.

- The first is the need for a graphical method for storing system requirement specifications and capturing the software requirements. A graphical method can contribute to overcoming a weakness of natural language in visualizing functional requirements by providing both an analytical synthesis of the requirements and, simultaneously, a means for verifying the various levels of detail of the requirements that are described in the SRS, or those that should have been included in it.
- The second is the need for a procedure standardizing the set of functionalities common to all systems.

To address these problems, an **I**ntegrated **M**easure for **F**unctional **R**equirements (IMFR) has been designed. The IMFR is based on the strengths of the following techniques:

- Graphical Requirement Analysis (GRA), proposed originally to integrate system/software functional requirements [2,3], translates the functional requirements from textual form to a logic-based graphical form.
- COSMIC-FFP, designed and implemented to measure the functional size of software, provides a procedure for describing a functional requirement based on a generic model of a software functional process. This includes identification of the sub-processes that must be designed to support input, process logic, output and interface data to, from and within the software.

2. Backgrounds and Related Work

2.1 Requirement Analyses

Descriptions of the functional requirements (what the system is supposed to do) are provided to the developers by the stakeholder and are usually prepared in natural language. However, as noted in one study [4], "*There is no known way in which all the required details of functionality can be extracted from natural language text with any degree of certainty*." Contrary to the views expressed in many books [5,6], "*functions cannot be deduced necessarily or exclusively from the use of verbs?*" Being able to obtain correct requirements and get them right first time has been a desire of software engineers, but there has been little available in terms of analytical tools to equip them with the adequate means to do so. There is, of course, even less available to enable them to visualize the requirement specifications, or the quality of such specifications.

History tells us that the greatest numbers of errors – and the errors that are most costly to fix – are generated at the earliest stages of development. Problems not found until the testing stage are at least 14 times more costly to fix than problems found during the requirements phase. The use of natural language to prescribe complex dynamic system functionality causes at least two severe problems: ambiguity and inaccuracy.

Managing the functional requirements is another critical issue: in most system development programs, since functional requirements are of a dynamic and volatile nature. As new requirements are added, or as existing ones are updated, deleted or modified, a management process should be in place to provide traceability and impact analysis to ensure that each of the changes is properly included in the system development process. At a minimum, continuous verification and validation procedures must be in place to ensure that stakeholder needs are met.

As with other human-designed activities, or processes, measurement quality is a challenging aspect of getting these activities or processes right. This is also a valid concern in the context of getting the requirements right. Currently, techniques proposed to measure the quality attributes in the requirements phase are mostly intuitive interpretations, based on experience and supported by project feedback; such measures are either indirect or, at best, nominally based. While defining and measuring the quality of functional requirements is critical for project scheduling and monitoring, there are no published or industry standards, or guidelines, for SRS measurement. These issues are investigated in this paper, and a proposal on how to address these issues with the SRS is presented.

In section 3, existing approaches to requirement quality attributes are summarized and the inconsistencies are discussed. This paper focuses on the challenges faced in measuring the quality of an SRS from three perspectives: (1) writing a requirement with correct functionality, (2) managing the volatility of requirements, and (3) measuring the quality attributes of requirements. To tackle these problems, we introduce an integrated model to provide a new solution to these challenges. This study also illustrates how IMFR captures critical aspects of functional requirements, such as ambiguous or incomplete requirements, and incomplete linkages from software requirements to system requirements, as well as to design and/or test cases.

2.2. Quality Attributes of Software Requirements

In order to implement a successful measurement program, project managers need well-defined criteria, valid measurement methods and reliable tools and analytical techniques to help them. If the requirements are ambiguous, incomplete or difficult to understand, then the risk of an unsatisfactory final product is increased. Functional requirements in the SRS should be traceable from the system requirements (or user requirements) to the software requirements document, through design and implementation and through test. As noted in IEEE 830-1998 [8], *"There is no tool or procedure that ensures correctness, but traceability makes this procedure easier and less prone to error."* Besides this, some quality characteristics of the SRS can be improved or measured while others cannot. For instance, it may be possible to improve consistency and correctness, but not completeness.

In addition to the various individual quality models proposed in the literature, there have been recent attempts at the standard level (either IEEE or ISO) to define a more extensive quality model. McCall's quality model [10] identifies traceability, completeness and consistency as being factors contributing to correctness. Boehm's quality model [7] gives completeness and consistency as sub-factors of RELIABILITY. On the other hand, correctness and consistency are two sub-factors , which affect MAINTAINABILITY in the McCall quality model. Furthermore, according to ISO/IEC 9126 [9], testability is a sub-factor of maintainability. In industry, the Software Engineering Technology Center of the National Aeronautics and Space Administration (NASA) [11] has selected five quality attributes for evaluating requirements quality in their applications (ambiguity, completeness, understandability, volatility and traceability). The Nuclear Regulatory Commission [12], in contrast, has defined the requirements quality attributes as traceability, consistency, correctness, completeness, verifiability, understandability and ambiguity.

These various viewpoints on the quality of requirements are summarized in Table 1. Such a table of the various approaches highlights on the one hand that there is not yet a consensus, and on the other hand that none of these models tackles all the quality issues identified by any of these models

Both Table 1 and Figure 1 illustrate that, in the current state of the art, there are gray areas about what should be measured to ensure quality requirements and how they should be measured. There are areas where the initial set of "core" attributes listed in Table 1 overlap, and this can be visualized in Figure 1, where the complex relationships between the quality attributes of software requirement specifications and of their software product are illustrated, simultaneously taking into account the information from the three quality models already presented, plus some information from other authors.

Table 1: The quality attributes of software requirements

Quality Attributes	IEEE Std-830 [8]	ISO/IEC-9126 Criteria/Sub-criteria [9]	Boehm Factor/-Sub-factor- [7]	McCall Factor/-Criteria- [10]	NRC NUREG [12]	NASA SATC [11]
Traceability	X	-	-	Correctness/ -Traceability-	X	X
Consistency	X	-	Reliability/ -Consistency-	Reliability & Maintainability & Correctness/ -Consistency-	X	-
Correctness	X	-	-	Correctness	X	-
Completeness	X	-	Reliability/ -Completeness-	Correctness/ -Completeness-	X	X
Verifiability	X		-	-	X	-
Understandability	-	Usability/ -Understandability-	Maintainability/ -Understandability-	Maintainability -Simplicity-	Style	X
Ambiguity	X	-	-	-	X	X
Volatility	Stability	Maintainability/ -Stability-	-	-	-	X
Modifiability	X	Maintainability/ -Change-ability-	Maintainability/ -Modifiability-	-	-	-
Testability	-	Maintainability/ -Testability-	Maintainability/ -Testability-	Test ability -Simplicity-	-	-

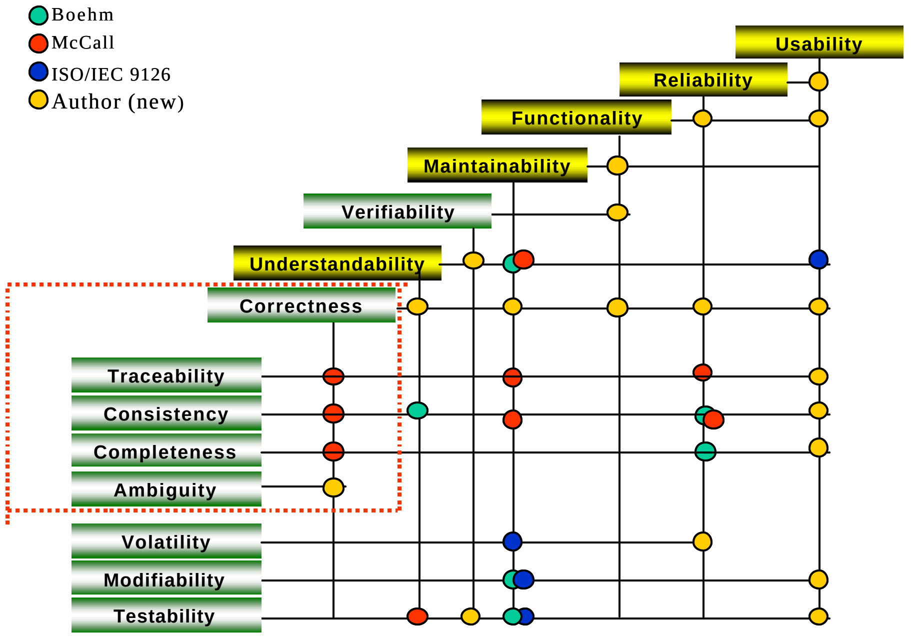

Figure 1. Analyzing the SRS quality attributes using the GRA Framework

2.3. Logic-based GRA Method [2,3]One way to avoid the ambiguity inherent in natural language is to write the SRS in a particular requirement specification language. A recently developed framework for building high-quality requirement specifications is based on the GRA method. GRA provides a graphical representation of functional requirements on a logic-based framework. Constructing a function with GRA is relatively easy for software developers, and the logic-based graphical method provides a precise, unambiguous basis for communication between developers and organizations. Procedures for constructing functional requirements with GRA have been summarized in the following four-steps:

Step 1: High-level requirements are collected from the system specifications and are grouped according to the goals and functions of the system. Subsequently, functional requirements are classified into two groups, describing the main functions and the support functions respectively. **Step 2:** Main and support functions are decomposed hierarchically into subfunctions. **Step 3:** The relationships in the hierarchies are represented by a connection between different nodes of a hierarchy or between nodes across two different hierarchies. The relations can be characterized as logical, physical or fuzzy (this is not to say that these are all the categories of relationships in a system). **Step 4:** The natural language functional requirements are translated into equivalent prepositional expressions using the definitions in step 3.

2.4. COSMIC-FFP [13]

Any measurement is based on the common acceptance of a model of a physical object (or of an abstract concept such as 'benefits'), on a shared way of representing it and then on the assignment of a numerical value according to specified and widely recognized scales. The measurement of functional requirements requires similar steps. For instance, the COSMIC-FFP functional size method recognizes two major steps for measuring requirements described in natural language: a mapping phase where the requirements are mapped in a very generic common model of the functional user requirements (FURs) of the software (and of its key concepts), followed by the assignment of numerical values according to simple measurement rules, once the mapping has been completed. COSMIC-FFP requires the execution of the following tasks for the software to be measured:

a. Identification of the software's functional boundaries:

b. Identification of the **functional process**: a functional process is a unique set of data movements (entry, exit, read, write) implementing a cohesive and logically indivisible set of FURs.

c. Identification of the triggering event; a triggering event occurs outside the boundary of the measured software and initiates one or more functional processes.

d. Identification of the data groups that pertain to this process for a specific **subprocess**.

e. Identification of the data attributes that pertain to this process for a specific subprocess; a data attribute is the smallest parcel of information, within an identified data group, carrying a meaning from the perspective of the software's FURs.

f. Assignment of the numerical values.

Tasks a to e deal with the mapping of the requirements to a generic model of software, and only step f deals with the assignment of numerical values. The output of the mapping tasks then provides a basis upon which to more easily apply the selected (preferred or mandatory) quality models to a set of standardized representations of the functional requirements

Table 2 Criteria of High-Quality SRS

Criteria	Description
Functional Requirements	
Definition of Functional Requirements	What the software is to do and how the software should respond to its environment.
Input and Output Requirements	These specify requirements for input and output of the software.
Software algorithms	Detailed description of the software algorithms.
Software data	The content of the information flows, with their formats and relationships.
External Interface Requirements	
System Interfaces	Each system interface should be listed, the functionality of the software required to accomplish the system requirements should be identified and the interface to match the system described.
User Interfaces	(1) The logical characteristics of each interface between the software product and its users, including such configuration characteristics as required screen formats, page or window layout, content of any reports or menus, availability of programmable function keys. (2) All the aspects of optimizing the interface with the person who must use the system. This may simply comprise a list of do s and don'ts relating to how the system will appear to the user. One example may be a requirement to offer the option of long or short error messages.
Software Interfaces	The nature of the information flows across software boundaries, where this information is found and under what conditions. This should be specified for the use of other required software products (e.g. data management system, an operating system or a mathematics package) and interfaces with other application systems (e.g. the linkage between an accounts receivable system and a general ledger system).
Hardware interfaces	What the software must do to transfer data across hardware boundaries, e.g. number of ports, instruction sets, etc)
Communication interfaces	The various interfaces with communications software should be specified, such as local network protocols, etc.
Constraints	
Software constraints	Imposed limits placed on the software and its simulation and responses.
Design constraints	Are there required standards to be met, e.g. implementation language, policies for database integrity, resource limits, operating environment(s), etc?
Software error conditions	What constitutes a departure from the norm, under which conditions does one occur and what action should be taken?
Software states	Stable modes which the software may assume, under which conditions and as a result of what actions.
Performance Requirements	
Time-related issues	Speed, response times
Accuracy-related issues	An SRS should contain a requirement relating to the accuracy of code predictions relative to the phenomena to be modeled.
Quality Requirements (non-functional requirements)	
Software standards	Those forms of representation and the content of the requirements demanded by the development organization and the user organizations.
Software quality attributes	The conditions that the software must meet in order to be considered fit for use in its intended application. These requirements specify operation of the software, such as portability, maintainability and other non-functional quality attributes

3. The Approach and Its Application (IMFR)

The IMFR proposed next is an integrated measurement approach designed to address both the correct definition of functional requirements and the measurement of quality in terms of quality attributes such as traceability, consistency, ambiguity and completeness. In addition, the GRA notation of the software functional requirements provides a visualization of functional requirement correctness. The COSMIC-FFP measurement rules and procedures are mapped onto the logic-based graphical representation of the FUR model of the software. Injecting the COSMIC-FFP measurement rules into the GRA framework provides, then, a structured procedure for identifying functional system requirements, thereby providing a means for verifying the clarity and the presence/absence of the requirements, or details of requirements, which should meet the broad consensus of a generic model.

The IMFR captures the concepts, definitions, interfaces and relationships – functional structure – between user functional requirements and their subfunctions, within a function and across functions. Building a measurable functional requirement process with the GRA method involves three well-known engineering approaches which have already been used for modeling and analyzing complex physical systems: (1) hierarchy theory, (2) the success/failure paradigm, and (3) Dynamic Master Logic [14]. The consolidation of the COSMIC-FFP measurement model and procedures into the GRA method provides the structural procedures for the IMFR measurement approach.

3.1 Build Measurable Functional User Requirements

Terminology: To avoid confusion, we have used the terminology defined in IEEE 610.12: A software module that performs a specific action is invoked by the appearance of its name in an expression; it? may receive an? input value and return a single value. When a function is decomposed, subfunctions can be identified. These definitions of function and subfunction correspond to the functional process/subprocess of the COSMIC-FFP measurement method.**Using Hierarchy Theory and Functional Modeling in the IMFR:** Frequently, complexity takes the form of a hierarchy, whereby a complex system is composed of interrelated subsystems which, in turn, have their own subsystems, and so on, until the lowest level of elementary component is reached. Hierarchic systems are usually composed of only a few different kinds of subsystems arranged in various combinations. Since only a finite number of basic parts constitute the building elements of a complex system, one only needs to know the common properties of these basic building blocks and their interactions with each other in order to describe the system. This also highlights the important role of class representation (or, more correctly, the class object), instances of which describe specific properties of parts of the system [14]. Since the parts of a system are functionally interrelated, FURs can be classified and grouped based on their objectives and purpose using hierarchy theory and functional modeling.

Using the Success/Failure Paradigm in the IMFR: There could be multiple support functions in a successful implementation of the main FURs, even though the user might not have a direct interest in them. Every high-level software functional requirement can be a function of many variables. When software functional requirements are defined for a specific application domain, many subfunctions and/or subprocesses could contribute to the success of the main FURs.

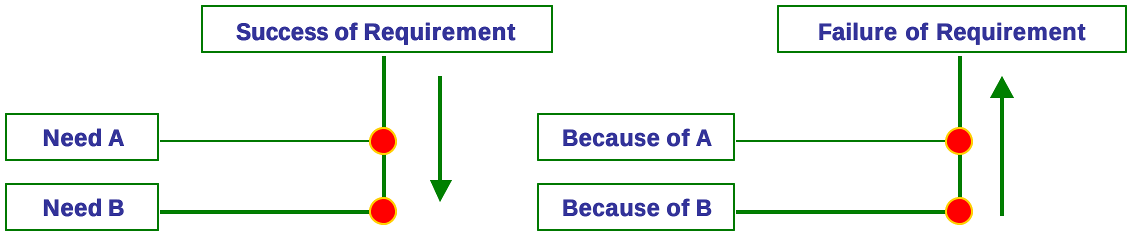

Figure 2 Success & Failure Paradigm

Using the GRA Framework for the IMFR: This involves a type of modular decomposition in which a system is broken down into components, which correspond to the system functions and subfunctions. When the hierarchical decomposition has been completed, subfunctions can easily be obtained. To achieve the low-level sub-functions, inputs from outside the boundary are necessary which are not user requirements, but rather system requirements. Those inputs could be physical variables (level, pressure, temperature) or outputs of another function/module (support function/module), or they may be the user's/operator's inputs. As illustrated in Figure 3, in many cases one input can be used to contribute to the fulfillment of more than one FUR or subfunction. Each node in the Figure represents I/O relationships.

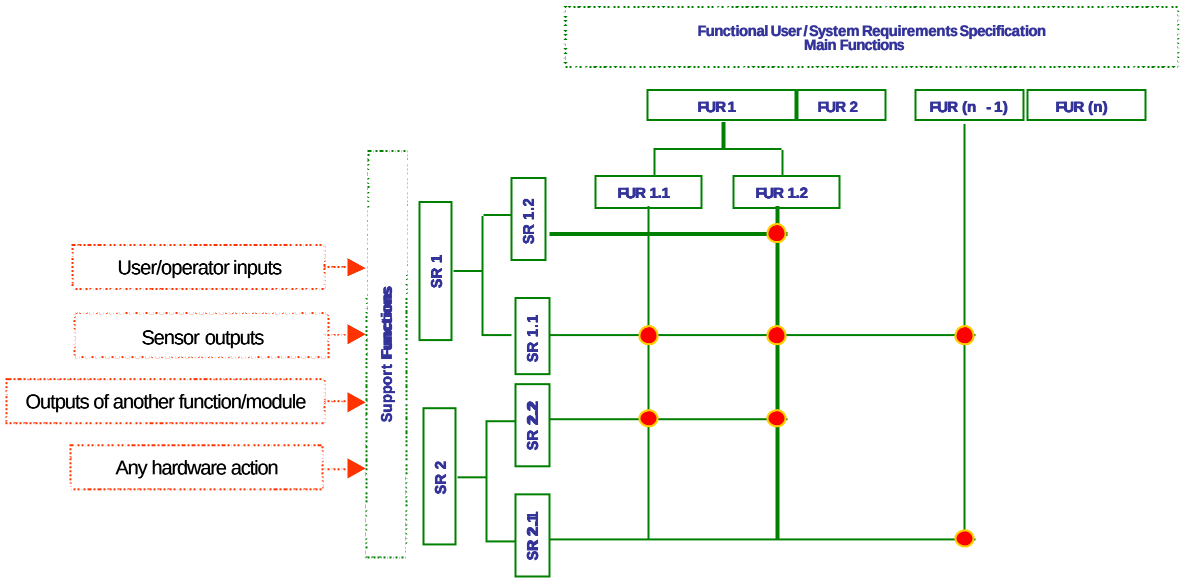

Figure 3 Definitions of Requirements with the GRA Methodology

3.2 Procedure of the IMFR Measurement Model

STEP 1: Identify COSMIC FFP boundary

- Define the purpose of each FUR - “What is it supposed to do?”.
- Apply *hierarchy theory* on all FURs to define a functional process.
- Apply the *Success/Failure Paradigm* on each FUR to identify each main and support function.
- Apply the *GRA framework* to define I/O relationships between the FUR and Support Functions.

STEP 2: Identify functional processes and subfunctional processes

- By applying the multilevel hierarchy theory, define all functions and subfunctions (main or support);

- By using the high-quality criteria on all requirements, define all input/output/trigger events/time issues of subfunctions and functions;
- By identifying the I/O relationships between each and every functional process.

STEP 3: Identify Write/Read/Entry/Exit Data Elements

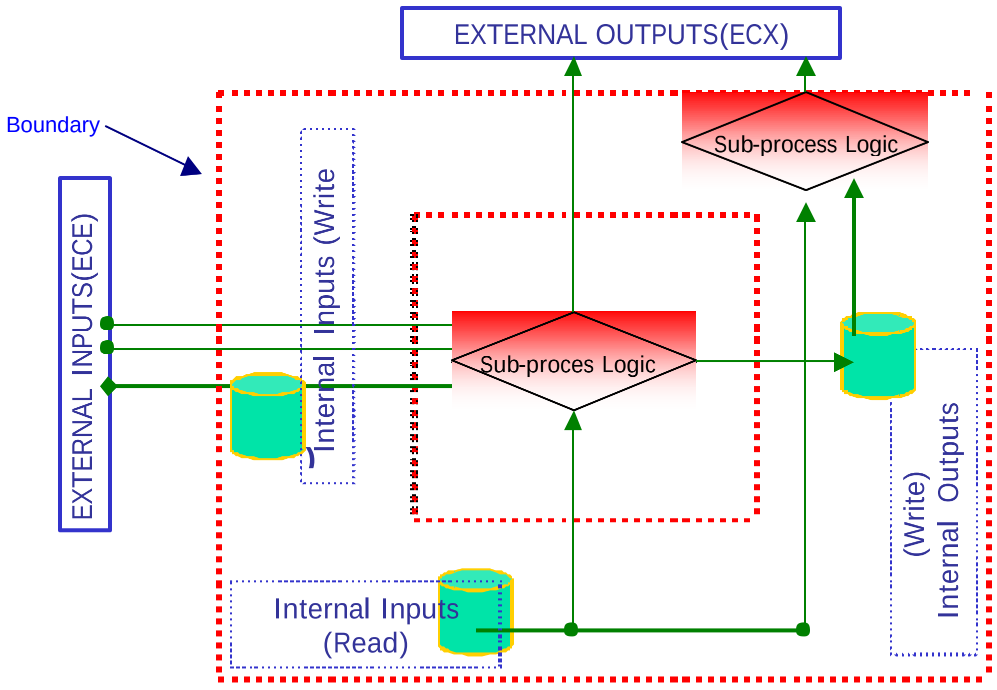

Figure 4 Definition of a sub-function/ sub-process into IMRF"Sub-proces" needs another s = " Sub-process"; also delete hyphen

4. An Example: Reactor Protection System Trip Function

The various reactor trip signals automatically open the reactor trip breakers whenever a condition monitored by the reactor trip system reaches a preset level. The implementation of the proposed model on the subfunction of Reactor Protection System Trip Functions (pressurizer water level control system) is demonstrated in this section as an example.

4.1 Building Correct Functionality with GRA

STEP 1 Definition of functionality: - "What is the function supposed to do?" Group them in order of their goals. The following information is taken from the Generic Westinghouse Reactor Protection (GWRP) System Specification [15]:

Reactor protection system functions are designed for:

1. *Monitoring the values of specific variables and comparing them to their respective set point,*
2. *Initiating reactor trip if the safe operating limits are exceeded. They will also initiate the engineering safety features if an accident occurs.*

Three pressurizer water level channels are used for the reactor trip. Isolated signals from these channels are used for pressurizer water level control.

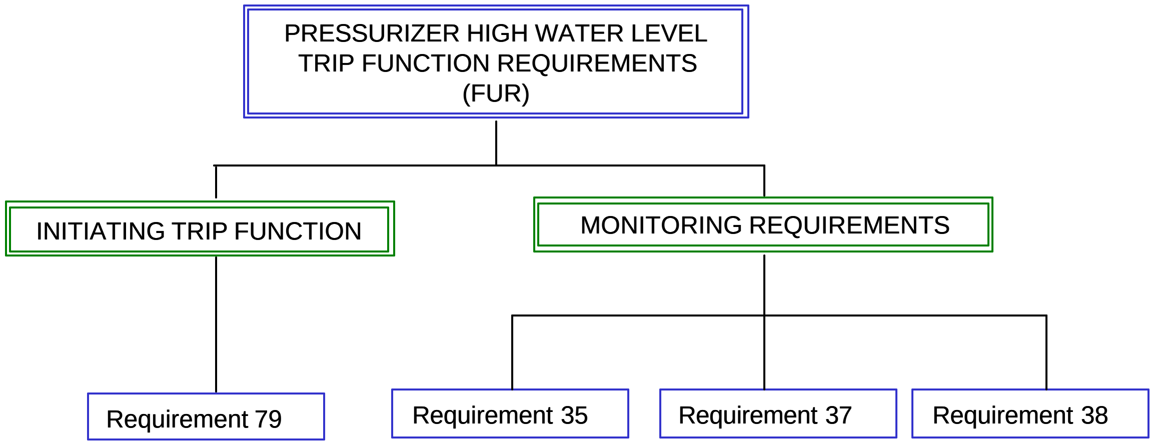

Figure 5 Identifying and grouping the FURs into hierarchy

STEP 2 Identify boundaries: As in defined in step 1, the functional requirements are classified into two groups: trip function requirements and monitoring requirements. These requirements are illustrated in Figure 5 using hierarchy theory. However, this information cannot provide for the concept of boundary, or the concepts of subprocess and process, as mentioned in COSMIC FFP. To fulfill the FUR successfully, five different inputs are necessary. One out of five inputs is provided by another requirement as an input to FUR 79 (output of FUR 87). Furthermore, FUR 87 has to be obtained from the output of FUR 86. The I/O relationships (represented by red dots) between FURs and support functions are illustrated in Figure 6.

Group A: High Water Level Trip Function (FUR 79):
Group B: Monitor Requirements (FUR 35, 37 and 38)

- FUR 35: All press-water levels and set points are displayed as percent total water level.
- FUR 37: A circular indicator labeled "P-7" is gray when P-7 is FALSE and green when it is TRUE
- FUR 38: A rectangular indicator labeled "TRIP" is gray when Trip is TRUE and red when it is FALSE

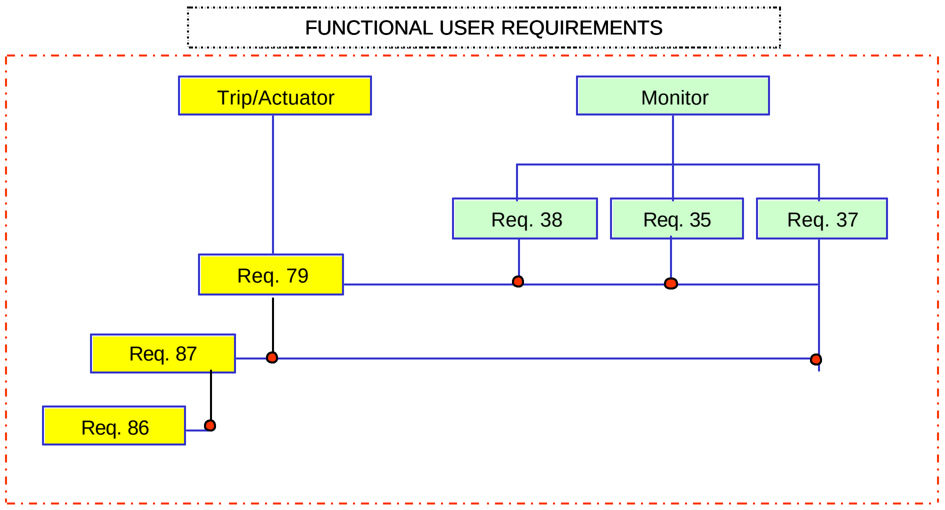

Figure 6 Identification of Boundary and I/O relations with GRA

STEP 3: Build functionality using the logic-based GRA framework: The functional logic of the high water level trip (FUR79) has been defined in the software requirements specification, as mentioned above. However, Figure 7 illustrates that the statement *"IF (2 out of 3x pressure level> high pressure level set point)"* has been interpreted in two different ways by two different analyzers. The GRA framework can precisely describe functionality to avoid such an ambiguous, inconsistent and incomplete definition of an FUR. Once the FUR has been built with GRA, verification of the FUR becomes much easier.

SRS of High Water Level Trip Function (FUR 79):

Inputs: (1) A-Press-Water-Level, (2) B-Press-Water-Level (3) GPress-Water-Level, (3) High-Press-Water-Level Set Point, (4) P-7, (5) High-Press-Water Level-Trip Set Point.

Output: High Water Level Trip Function

Logic: IF (2 out of 3x press-level > high press-level set-point) AND (P-7= FALSE)

```
    THEN high-press-level-trip = FALSE;
    ELSE high-press-level trip = TRUE;
    END.
```

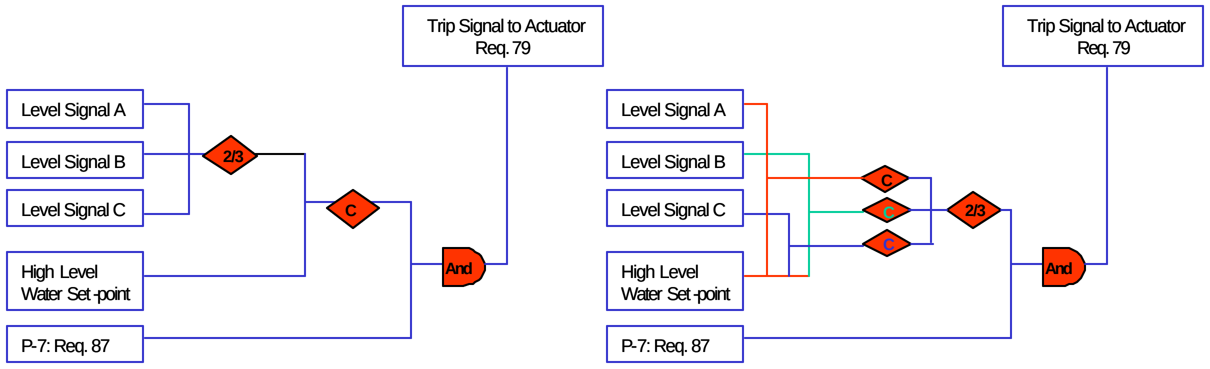

Figure 7: Graphical logic-based representations of High Water Level Trip (FUR 79), by two different analysts from the same FUR description – containing one ambiguous description

The functionality of FUR 79 (a sub-process), as is shown in Figure 7, can be interpreted in two different ways by different software designers. GRA is a method by which these types of ambiguities in functional descriptions can be avoided and/or captured.

STEP 4: Apply "Success/Failure Paradigm" in the GRA framework to define the I/O relationships between subfunctions/support functions and FURs

To succeed, the Trip Function (FUR 79) requires five inputs. Three level signals are obtained from level sensor outputs. Also, a set point has been defined, though the output of permission 7 is provided by another function of the system. For *p-7 (FUR 87)* to succeed, the output of *p-10* (FUR 86) is necessary. The functionality of P-10 is built with GRA and illustrated in Figure 8. It is clear that FUR 79 needs FUR 87 and FUR 86 to achieve its functionality. It will then be very easy to identify input/outputs, subfunctions (subprocesses), logic and the relationships between them in the GRA framework, as shown in Figure 8.

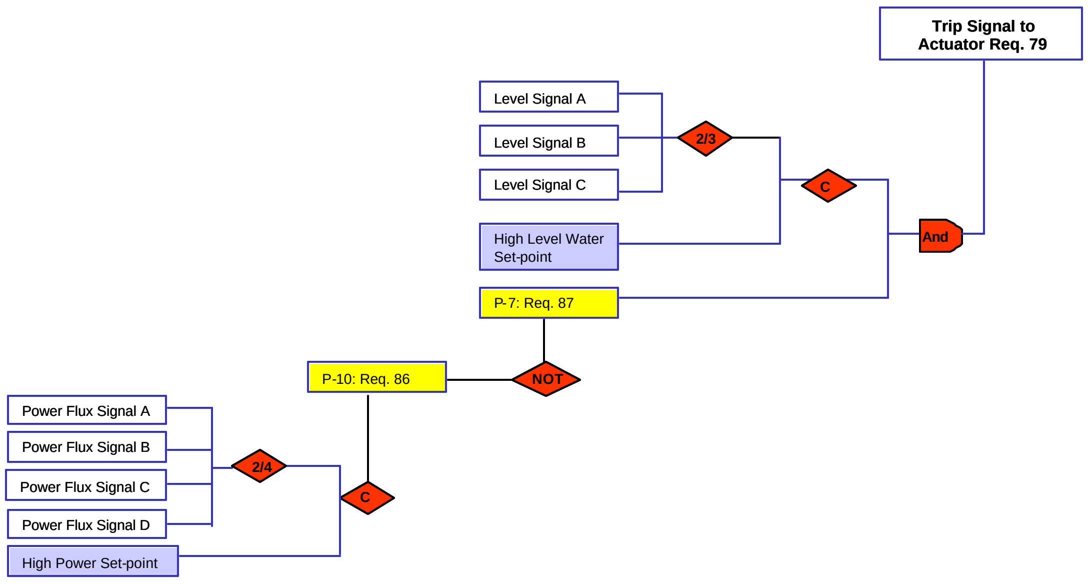

Figure 8 Success/failure paradigm of trip signal (FUR79) in the logic-based GRA framework.

STEP 5: Implementation of the IMFR Measurement Model on the GWRP "High Water Level Trip Function"

- TRIGGER (TIME): The software samples every 100 milliseconds in real-time. Real time is accessed by the computer's internal clock. (FUR 1)
- EXTERNAL INPUTS: LA, LB, LC, FA, FB, FC, FD (7 FFP)
- EXTERNAL OUTPUTS: trip function (ON/OFF), monitor requirement {(values of LA, LB, LC), (position of P-7), (position of trip function)} (4 FFP)
- INTERNAL WRITE: External inputs {LA, LB, LC, FA, FB, FC, FD} and subprocess outputs {p-7, p-10, Trip} (10 FFP)
- INTERNAL READ: L-SP, F-SP, LA, LB, LC, FA, FB, FC, FD (10 FFP)

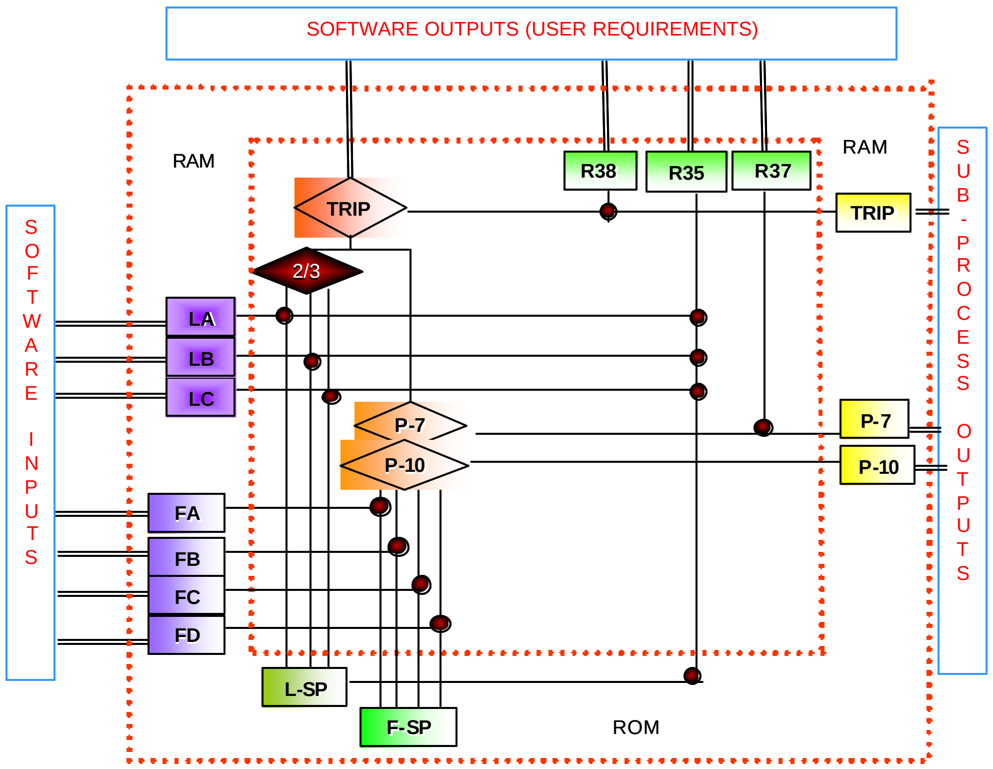

Figure 9 COSMIC-FFP generic model of software (for functional size measurement) in the GRA method.

4.2 Measuring Functional Correctness with the IMFR

The architecture of the proposed IMFR model captures at least[1] the following quality attributes of software functional requirements: ambiguity, consistency, completeness, traceability.

Completeness: If any input/output of logic has been forgotten in the SRS, the IMFR can catch them during the decomposition process. Building functionality using the architecture of the IMFR requires, and enforces, the specification of the characteristics of each FUR, such as the goal of the particular functionality, inputs, outputs, logic, trigger, boundaries and interfaces. Missing components and characteristics of functionality can be detected with the generic built-in representation of software functionality within the IMFR.

Consistency: If two FURs are in conflict with each other in the SRS, because of the tree and lattice structure of GRA, the IMFR can detect conflict requirements easily. For instance, one requirement might have specified that the program will add two inputs and another has specified that the program will multiply them. One requirement might state that *A* must always follow *B*, while another requires that *A* and *B* occur simultaneously.

Ambiguity: An SRS is unambiguous if, and only if; every requirement stated therein has only one interpretation**.** In the case where a term used in a particular context could have multiple meanings, with the logic-based graphical language of the requirements, the IMFR can capture ambiguity very easily, as demonstrated in the high water level trip function example.

Traceability: The architecture of the IMRA provides the help needed to follow each function in the requirements phases, from the design through to implementation. When a requirement in the SRS represents a derivative of another requirement, both forward and backward traceability are provided by the IMFR. In addition, the relationships between different phases of the development life cycle can be identified in the same architecture.

Interfaces and boundaries: The hierarchical decomposition technique is used to decompose each FUR into independent modules with clearly defined interfaces and boundaries.

[1] The method can be useful for other quality attributes as well, such as maintainability (testability, modifiability, volatility). However, this study focuses only on the correctness attributes and not on maintainability.

Verifiability: An SRS is verifiable if, and only if; every requirement stated therein is verifiable [IEEE 830-1999]. When an FUR is built with the IMRA, it presents all the necessary characteristics of the function, including subfunctions and support functions. For example, the Trip Function (req.79), which is one FUR, is illustrated with all necessary components in Figure 8. The complete pressurizer high water level trip function requirements are also visualized in Figure 10. Users, customers and developers can agree on the same understanding of functionality in a visual way. Verification of functional correctness can be provided when a tool is available for automation.

Functional representation and size measurement are an integral part of the proposed model.

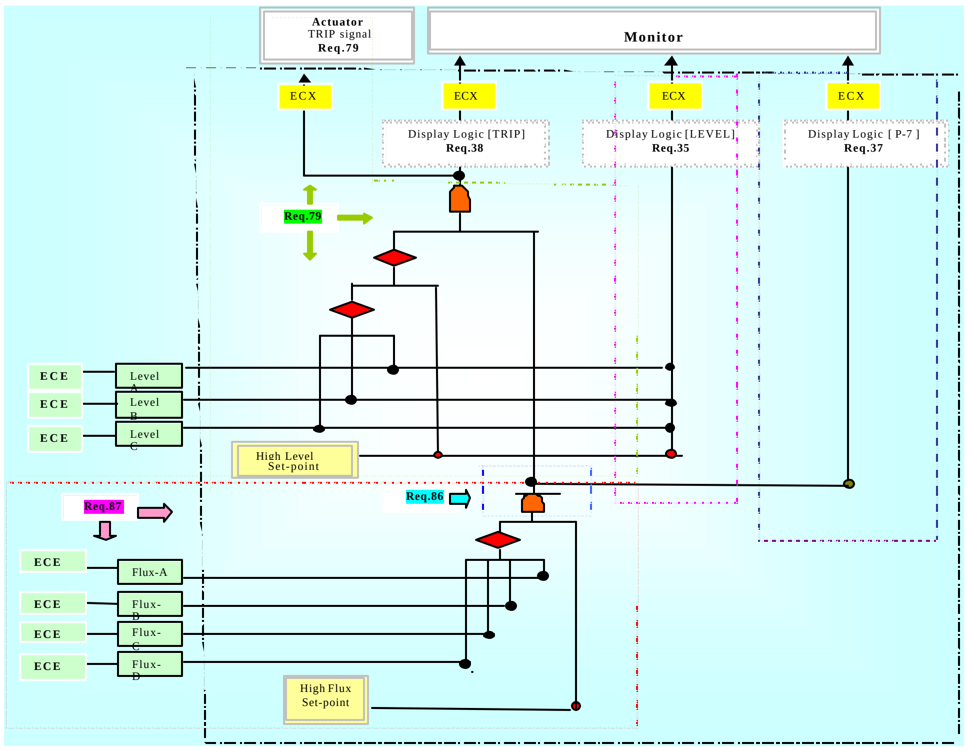

Figure 10

5. Conclusions and Future Work

In this paper, three different perspectives of Software Requirement Specification (SRS) have been discussed: (1) the quality attributes of SRS from a measurement perspective, (2) the quality criteria of a good SRS from the quality assurance perspective (inspection/review/audits), and (3) requirement analysis methods from the requirements engineering and management perspectives. It has been illustrated that, even when using any of these, the description of an SRS can still be incomplete or ambiguous. To address this issue, the IMFR approach has been designed with the following characteristics:

1. The model provides an especially efficient and accurate way to specify functional requirements.
2. In the model, requirements are expressed using a logic-based graphical technique which makes it possible to avoid the ambiguity inherent in natural language as well as to detect ambiguous FURs.
3. The model provides a mapping from inputs to outputs into a multi-level detailed system and software functionality. It can help identify and define various modules of software or demonstrate the intended functionality of any FUR.
4. The modular structure can be built into, and verified in, the architecture of the model. The model provides the interconnections between module, functional block, functions and subfunctions and/or subprocesses.
5. The model captures most of the characteristics of a high quality SRS, and the criteria that are recommended in the literature and in various standards can be built into the architecture of the model.
6. The model provides a means by which to verify clarity and its presence/absence.

References

1. IEEE Std 730.1-1998 IEEE Standard for Quality Assurance Plans.

2. Kececi, N., M. Modarres and C. Smidts. "System Software Interface for Safety-Related Digital I&C Systems", *European Safety and Reliability – ESREL'99 Conference*, TUM Munich- Garching, September 13-17, 1999.

3. Kececi N, M. Li and C. Smites, C. "Function Point Analysis: An Application to a Nuclear Reactor Protection System," *International Topical Meeting on Probabilistic Safety Assessment –PSA'99*, Washington, DC, August 22-25, 1999.

4. Hennell M.A.1987. *Requirements, Specification and Testing. Software Reliability Achievement and Assessment, Edited by B. Littlewood.* Blackwell Scientific Publication.

5. DeMarco T. 1979. *Structured Analysis and System Specification.* Prentice-Hall.Jones C.B. (1980) *Software Development.* Prentice-Hall.

6. Jackson M.A. 1983. *System Development*. Prentice-Hall.

7. Boehm, B.W., J.R. Brown, J.R. Kaspar, M. Lipow and G. Mac Cleod. *Characteristics of Software Quality. Amsterdam*: North Holland. 1978.

8. IEEE 830-1998, "*IEEE Guide to Software Requirements Specification*".

9. ISO/IEC 9126 "*Information Technology Software Quality Characteristics and Metrics*"; Part 1: Quality model, Part 2: External metrics, Part 3:Internal metrics.

10. McCall, J.A., P.K. Richards and G.F. Walters. *Factors in Software Quality*, vol. 1,2 and 3, AD/A-049-014/015/055. Springfield, VA: National Technical Information Service, 1977.

11. NASA/ SATC Linda H Rosenberg, T.K hammer, L.L. Huffman "Requirements, Testing, and Metrics" NASA SATC Software Engineering Technology Center http://satc.gsfc.nasa.gov.

12. NRC/NUREG-0800: HICB-BTP-14, "*Guidance on Software Reviews for Digital Computer-Based Instrumentation and Control Systems*". U.S.A. Nuclear Regulatory Commission 1999.

13. Abran, A., Desharnais, J.-M., Oligny, S., St-Pierre, D., Symons, C., COSMIC FFP – Measurement Manual version 2.1, Montréal (Canada), May, 2001. http://www.lrgl.uqam.ca/cosmic -ffp

14. Modarres, M. "*Functional Modeling of Complex Systems Using a GTST-MPLD Framework*". Proceedings of the 1st International Workshop on Functional Modeling of Complex Technical Systems, Ispra, Italy 1993.

15. Westinghouse Technology System Manual, United States Nuclear Regulatory Commission Technical Training Center.Rev.0690 Vol.2, p. 12.

16. IEEE 610.12-1990, "*Standard Glossary of Software Engineering Terminology (ANSI)*".