Requirements Completeness

Conference Paper *in* INCOSE International Symposium · June 2004

DOI: 10.1002/j.2334-5837.2004.tb00546.x

Requirements Completeness

Prepared by the Requirements Working Group of the
International Council on Systems Engineering
For information only; not approved by INCOSE Technical Board
Not an official position of INCOSE.

PROJECT TEAM
Ronald S. Carson, Ph.D.
ronald.s.carson@boeing.com
Boeing Integrated Defense Systems
P.O. Box 3707, MC 8E-94
Seattle, WA 98124-2207, USA

Erik Aslaksen
George Caple
Paul Davies
Regina Gonzales
Ron Kohl
Abd-El-Kader Sahraoui

Abstract. The INCOSE Requirements Working Group has attempted to define a process and methodology for producing a *complete* set of requirements. We first carefully define the problem statement and terms. Various approaches to completeness are reviewed, and a formal process is proposed to quantify completeness.

Introduction

The systems engineering process is an iterative process, with each cycle of the iteration increasing the level of detail of the design. The process operates over a number of life-cycle phases, typically (INCOSE SE Handbook 2000, Section 3):

- Definition
- Analysis
- Design
- Implementation
- Verification
- Operations and Maintenance
- Disposal

The first of these phases results in a set of requirements, which form the basis for the following phases. Requirements are therefore the critical link in the whole systems engineering process.

The Requirements Completeness Project Group within the Requirements Working Group (RWG) began this task in Autumn 2002, and defined its mission as follows:

> Develop and validate a methodology that can produce a *complete* set of requirements and that can determine the completeness of a set of requirements.

Because requirements drive the system design and operation (at least we typically hope that is the case), it is apparent that the requirements should be complete. (Kar and Bailey 1996) among others note the importance of having complete requirements. As (Wiegers 1999) notes, however, "Missing requirements are hard to spot, because they are invisible." As (Hooks and Farry 2001) note, "The second largest class…of requirements errors…was omissions."

Initially it was believed that with sufficient review and audit a set of requirements could be concluded to be complete, although there was no formal method or proof involved. This is the functional equivalent to "inspecting quality in", a methodology rejected by those involved in producing high-quality goods and services. The quality must be designed and built into the product. If we consider "high-quality" requirements to have the property of completeness, what is the process to get to that condition? How can a set of requirements be confidently produced as a *complete* set?

As (Carson 1998) notes:

> (Mar 1994) identifies 5 characteristics for establishing requirements completeness: "(1) all categories of requirements (functional, performance, design constraints, interfaces) ... are addressed, (2) …all responsibilities allocated from higher-level specifications are recognized, (3) …all scenarios and states are recognized and no To-Be-Determined (TBD'S) exist..., (4) all assumptions are documented..., and (5) …the requirements conform to appropriate, and [have] unique or ambiguous terminology defined." Among these the most difficult to ensure are (1) "all categories... are addressed", and (3), that "all scenarios and states are recognized."

These "most difficult" items constitute the issue of *completeness.*

What is a Requirement?

Preparatory to defining *completeness*, we first examine what is meant by a *requirement*. (Harwell et al. 1993) proposed that

> If it mandates that something must be accomplished, transformed, produced, or provided, it is a requirement – period.

Or equivalently (IEEE 1220, 3.1.33):

> A [requirement is a] statement that identifies a product or process operational, functional, or design characteristic or constraint, which is unambiguous, testable or measurable, and necessary for product or process acceptability (by consumers or internal quality assurance guidelines).

Not surprisingly, this approach is also indicated in the United States Department of Defense standards (MIL-STD-961E, 5.8):

> This section shall define the requirements that the entity must meet to be acceptable…
> c. Only requirements that are necessary, measurable, achievable, and verifiable shall be included.
> d. Requirements shall be worded to provide a definitive basis for acceptance or rejection.

This position is also confirmed in at least some of the software literature (Robertson and Robertson 1999):

> A requirement is something that the product must do or a quality that the product must have.

These definitions clearly convey the notion of a *mandatory* provision. But not all authors support this position. In particular, (Gabb et al. 2001) use "requirement" in a non-mandatory sense:

A requirement is an expression of a perceived need that something be accomplished or realized.

In this paper we are concerned with completeness as regards the *mandatory* provisions for reasons that will become evident.

Relation to Other Requirements Attributes

Requirements can be characterized by a number of attributes. Some of those found in the literature are listed in Figure 1[1]. Of these attributes, this paper is concerned with a simple one – completeness. It is an attribute that that has proven exceptionally difficult to define in a form that is both operational and generally accepted.

Given the large number of attributes listed in Figure 1 it is only natural to ask if there is some form of ordering among them, and where in such an ordering the attribute of interest – completeness- might sit. To answer this we need to have a quantitative characterization of these attributes, and in the following we introduce a grouping of attributes according to a three-dimensional scheme.

The **first** grouping of the properties is into two subgroups:

a. Those that refer to the requirement itself

b. Those that refer only to the formulation of the requirement (how the requirement is stated).

In the second subgroup are accuracy, conciseness, conformance, and unambiguousness; the rest are all in the first subgroup.

The **second** grouping is according to how the attribute is determined:

a. There exists a process for determining (and testing) it. This is true for accuracy, boundedness, conciseness, conformance, consistency, orthogonality, unambiguousness, and verifiability.

b. The value of the attribute is estimated. This applies to affordability, complexity, and risk.

c. The value is assigned by the stakeholders. This applies to correctness, criticality, priority, and *completeness*.

d. The value is intrinsic. This applies to class and level.

The **third** grouping is according to the *context* in which the attribute is (or should be) used:

a. In the context of a single requirement (accuracy, class, conciseness, conformance, correctness, criticality, level, unambiguousness, verifiable).

b. In the context of the requirements in a set (affordability, boundedness, and *completeness*).

c. In the context of the relations between the members of a set (consistency, orthogonality).

d. In the context of the process of fulfilling the requirement (complexity, priority, risk).

[1] Note that in Figure 1, the "attribute" is listed without prejudice; *i.e.*, the degree of "affordability" is the value of an attribute of a requirement. That a requirement must be "attainable" or "cost-effective" is a characteristic of a "good" requirement. "Set" vs. "Individual" refers to whether the attribute is assessed in the context of a single requirement or the total *set* of requirements.

Attribute	Application	Comments
Accuracy	Individual	*Level of detail*
Affordability	**Set**	*attainable* (Kar and Bailey 1996), *cost* (Jones 1998)
Boundedness	**Set**	*scope identified* (IEEE 1233)
Class	Individual	(Gabb et al. 2001) Functional, performance, direct (the latter defined in (Jones 2002)), also *boundary condition* (Kar and Bailey 1996)
Complexity	Individual	Of fulfilling or (more often) of verifying
Completeness	**Set**	(Kar and Bailey 1996), (Carson 1998), (Hooks and Farry 2001, p. 63).
Conciseness	Individual	*straight*
Conformance	Individual	*standard constructs* (Jones 1998), conformance to standards
Consistency	**Set**	(Schneider & Buede, 2000), (Hooks and Farry 2001, p. 63)
Correctness	Individual	(Schneider & Buede, 2000)
Criticality	Individual	Also *importance* (Jones 1998), *compliance* (Kar and Bailey 1996)
Level	Individual	Primary, derived (Harwell 1993)
Orthogonality	**Set**	*normalized* (IEEE 1233)
Priority	Individual	The order in time of fulfillment (Gabb et al. 2001)
Risk	Individual	*certainty* (Jones 1998)
Unambiguousness	Individual	*complete, standalone* (Kar and Bailey 1996), refers to both semantics and syntax
Verifiability	Individual	*Testable* (Schneider & Buede, 2000)

Figure 1. Requirements Attributes.

According to Figure 2, the properties are characterized by three parameters, one for each group, with the first parameter taking on one of two values (a or b), and the other two parameters taking on one of four values (a, b, c, or d), so that there are a total of 32 different combinations.

The attributes with identical parameters can now be grouped, *e.g.,* accuracy, conciseness, conformance, and unambiguousness all belonging to the group (b, a, a). *Completeness* is in a group of its own (a, c, b). Thus, it is not possible to deduce the nature of this attribute by reference to other attributes in the same group; we need to go back to basics. A necessary (but insufficient) condition for completeness is that each requirement has all the attributes of a "good" requirement (Kar and Bailey 1996).

What is Completeness?

Normally we think of something being *complete* when it lacks nothing, or requires nothing more in order to be fit for use or serve its intended purpose. The Merriam-Webster dictionary defines *complete* as "having all necessary parts, elements, or steps," which is consistent with our intuitive understanding.. For a set of requirements, this means that there are no other requirements necessary for the set of related requirements to fulfill their collective purpose or mission of defining what a given system must be and do, with respect to some specified context.

If we attempt to apply this definition to requirements, how do we determine what is "necessary"? And who decides?

Attribute	Group 1	Group 2	Group 3
Accuracy	b	a	a
Affordability	a	b	b
Boundedness	a	a	b
Class	a	d	a
Complexity	a	b	d
Completeness	a	c	b
Conciseness	b	a	a
Conformance	b	a	a
Consistency	a	a	c
Correctness	a	c	a
Criticality	a	c	a
Level	a	d	a
Orthogonality	a	a	c
Priority	a	c	d
Risk	a	b	d
Unambiguousness	b	a	a
Verifiability	a	a	a

Figure 2. Placing "Completeness" in the Framework of Groupings

If, as some argue, the purpose of a system is what it does (POSIWID), there is no hope of determining completeness until the system is operating, since it doesn't "do" anything until then. We cannot afford to wait that long!

On the other hand, if we take the position that human-designed systems are created with a purpose to accomplish a mission or achieve a goal, we can use these features to create a context for determining completeness. So we can begin to define *completeness* in terms of *fitness for purpose* (the *degree* to which it satisfies the needs, goals, and/or mission of the system). But who gets to decide? And what metrics will be used?

Determining the stakeholders. A number of papers (Gonzales 2001) have discussed requirements elicitation and stakeholder identification. Using the EIA-632 definition, a stakeholder is "An enterprise, organization, or individual having an interest or a stake in the outcome of the engineering of a system."

Thus, the system in its environment must be examined to determine who are the stakeholders. And this must be performed across all life-cycle phases (development through disposal). What we are attempting to identify are the features of the system that someone cares about. The customer or sponsor cares that it "works", that it accomplishes some purpose. Others will care about the unintended or undesired consequences of the system development or operating behavior.

Examining the proposed system in context of its life-cycle provides a simple means to determine stakeholders: anyone or any organization involved with or affected by the system lifecycle is a stakeholder. This can be conducted as a thought experiment, with the Systems Engineer playing the role of others who might be concerned but are not represented. (A simple shortcut to determine whether a requirement truly is mandatory is to attempt to delete that particular requirement and observe whether anyone objects.)

In addition, two specific stakeholders must be addressed. The first is "nature," which although is not a person cannot be ignored, in the sense that physical laws cannot be violated. The second are stakeholders who are not allowed to approve the requirements. An example would be the future generations who do not exist today, and another of stakeholders that do not have legal standing (the party has no right to participate). These individuals may lack the ability to respond, or may have more important issues currently, but could be impacted by a new system in the long run. Some systems are eventually approved by redefining the stakeholder set until the stakeholders that cannot comment are identified. This is sometimes the strategy for systems that

have “not in my backyard” impacts. Such stakeholders and their requirements may or may not be included; it is a choice on the part of the organizations and people chartered with developing and producing the system. But identified stakeholders who are excluded from participation create a risk for future activity.

All stakeholders (including the developer, one of whose jobs it is to clarify, integrate, and ensure completeness of requirements) bring their requirements to the system development process. Satisfying their needs by capturing their requirements is a necessary but not sufficient condition for completeness of a set of requirements.

Definition of Completeness. It is evident from the forgoing that there is no absolute measure of completeness (Carson 1998, Carson and Shell 2001). We must define completeness with respect to some measure. In particular, a given set of requirements may well establish the requirements for a specific system, yet be totally inadequate (incorrect and/or incomplete) for another system. How do we determine which is which?

As can be seen in Figure 3, the *context* of the system development activity includes stakeholders and various environments. A set of requirements appropriate for one set of stakeholders and environments could be quite different, *even if the missions of the systems are identical.* For example, there is no reason to expect that requirements for two automobiles would be identical if one were designed for European highway travel and another for off-road desert use, even if both sets of requirements could be established as “complete”. In addition to the obvious differences in environments, there may be differences in the operating concepts as well (*e.g.*, the need for more on-board fuel; provision for extra water). The stakeholder sets for each would also be quite different.

The context is actually two-fold, as is indicated in Figure 3. First, the need for developing or determining requirements completeness may occur in the *absence* of any higher-level set of requirements, for which mathematical techniques involving decomposition and traceability are not applicable. Thus, completeness of the highest-level “system” requirements is the most difficult problem to address. In this instance we are faced with the less-well-defined condition of the absence of a complete source of requirements (and the genesis of this RWG task). This aspect of requirements completeness will be the focus of the remainder of this paper.

In contrast, the second possible context is that there already exists a provably *complete* set of higher-level requirements from which a next-lower-level set of requirements is to be derived. In this second case, *completeness* can be established by allocation and traceability: have all requirements from the complete set been allocated? Is every requirement in the next-lower-level set of requirements derived from and therefore traceable to the source? Under these conditions the resulting set of requirements can be demonstrated to be complete (*i.e.,* there are no “missing” requirements, since all source requirements have been demonstrably allocated).

For example, if we measure completeness of software requirements in terms of external consistency the criteria for completeness would be that the set of software requirements satisfies the allocated system requirements; no more, nor less. In this context the notion of completeness must also include fitness for purpose; that is, lower level development and the eventual realization of product. For software, the derived requirements for the software must be sufficient to ensure that eventual closure or convergence on a realizable solution is possible. This can be applied recursively for a system, subsystems and components.

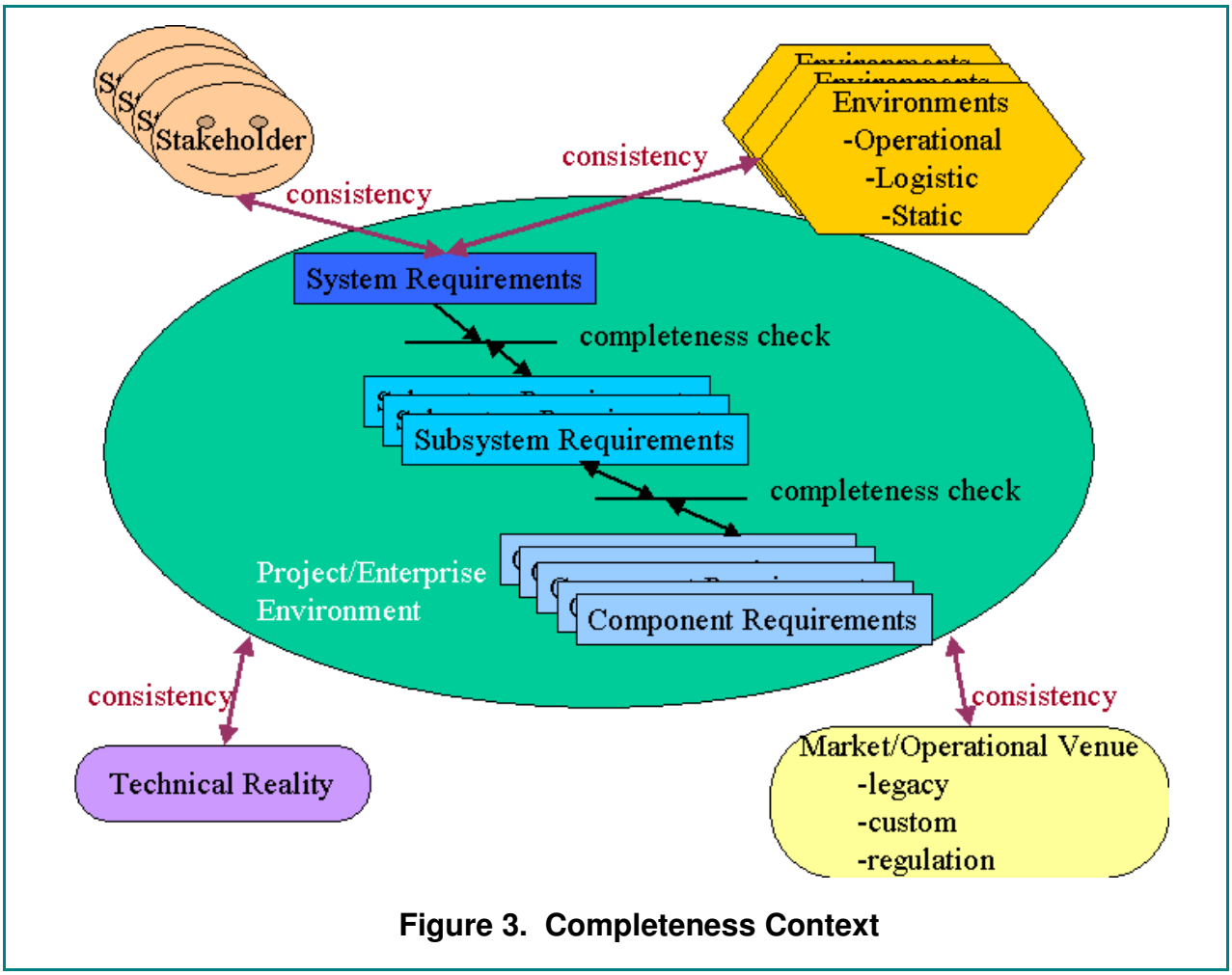

Figure 3. Completeness Context

Definition. For the purposes of this paper, then, we adopted the following definition:

A set of requirements is ***complete*** if and only if all stakeholders approve of the set of requirements.

The appeal to stakeholders should be apparent because of the context dependency we have attempted to establish. This does not mean that the process cannot be rigorous; we later show how such a process can be quite mathematically rigorous.

It follows that the *measure* of completeness is the *degree* to which all requirements in the set are approved by all stakeholders. Each stakeholder must be satisfied with the *set* of requirements, not just those for which he/she might be considered the "primary" stakeholder. Could all stakeholders approve the requirements and yet the set be incomplete? By definition, no. Could the stakeholder set itself be incomplete? Yes, and this is a risk in any system development activity. Could the approved set of requirements be inadequate for some specified purpose? Yes, but this would be a purpose different from the mission context in which the set of requirements was derived (see the discussion regarding the example of automobile missions above).

Does this definition mean that the requirements can be *complete* in abstraction from a system context, *i.e.*, can a group of stakeholders "approve" an incomplete set of requirements by failing to address the obvious system interfaces? The intent of the definition is that every stakeholder (including the developer) brings to the requirements development process some set of external interfaces or characteristics which are necessary from the vantage point of one or more stakeholder, *e.g.*, necessary for correct operation, necessary for durability, safety, certification, etc. It is not the *presence* of the stakeholder, per se, in the approval process, but that they approve the *content* of the set of requirements as adequately addressing the system context. The degree of approval of the set of requirements by the stakeholder group should not be a referendum on likes and dislikes (content, format, or semantics), but rather on whether the set of requirements addresses the needs for the system.

From the forgoing it should be apparent why it is impossible to attempt to establish "completeness" with non-mandatory provisions (or "perceived needs") as possible "requirements". Failing to formalize a "perceived need" as a true "requirement" or a feature that would be "nice to have" (sometimes known as a "desirement") leaves the developer in the area of uncertainty and an inability to make effective progress. There is no basis for determining when such "requirements" are complete, because there are no constraints on the problem of satisfying "perceived needs".

The commercial market place typically deals with these issues by making an informed assessment of the "requirements" for a project, based on market assessment, schedule, and financial resources. The primary feedback mechanism into the product life-cycle is customer response in terms of sales volume and post-sales comments on the features.

Survey of Techniques for Completeness

With few exceptions the general inclination has been that completeness is a probable result of following a process consisting of a combination of templates, checklists, stakeholder involvement, and lots of hope. But few papers have demonstrated processes which are sufficient to confidently provide complete requirements according to the above definitions. In this section we briefly summarize various techniques that have been offered.

Several approaches to requirements completeness have already been noted in some of the references, and fall generally into a few categories:

- A template or checklist approach, *e.g.*, Volere (Robertson and Robertson 1999), (Gabb et al. 2001), also military standard document templates. The theory is that addressing all elements of the checklist will ensure completeness.
- Requirements-elicitation basis. Approaches that involve users, *e.g.*, QFD and prototyping, can be effective in gaining at least partial stakeholder involvement and buy-in.
- Use-case and other functional analyses based on the mission and concept of operations which establish what the system must *do.* While useful for functional requirements these approaches do not typically address design constraints, although some approaches may include some environmental considerations (*i.e.,* "under what conditions is a specific requirement applicable?").

- Review, review, review (Grady 1993), until no one can think of anything else. Completeness is established by BOGGSAT[2].
- Context analysis: stakeholder identification and interface quantification. This is a mathematically based formal approach (next section).

These approaches are briefly described.

Templates and Checklists. Volere/Military Standards. The "Volere" process (Robertson and Robertson 1999) provides both a process as well as checklists and document templates. The method is highly interactive with stakeholders, which helps ensure completeness. MIL-STD-490 appendices (superseded by MIL-STD-961) provide specification outlines indicating recommended topics for individual paragraphs. (Hooks 1993) provides a list of requirement topics that should be checked. (Hooks and Farry 2001) note that "a standard specification format will reveal omissions" and "prevent loss of requirements."

QFD. QFD (INCOSE Handbook, 2000, Appendix A) has been used in engineering for many purposes such as eliciting requirements, comparing design characteristics against user needs, and ranking manufacturing methods. The inputs to the requirements QFD process are user needs, so it is essential that users participate. The user community participation should be as wide as possible (such as actual operators and maintainers in the field) to ensure that all aspects of user need are captured. Both 'essential' and 'nice-to-have' needs should be captured after thorough discussion. The QFD process attempts to correlate "user needs" with "system requirements". Each user need is then rated on a scale (*e.g.,* 1-5) to provide the system designer an understanding of the importance of each user need. Other methods, such as Analytic Hierarchy Process (AHP) can be used to prioritize needs. A key advantage of QFD compared with the other approaches is that the topics are not limited to functional requirements.

Prototyping. Prototyping helps the user and developer discover the functional requirements and user interface constraints of the system as a member of the user community interacts with elements of the system. Alternate scenarios can be discovered, potentially uncovering "what if?" requirements. This can help to form a more complete set of requirements by helping users understand what they want. The closer we can get to "real" use and the more experience the users have with the system to be deployed, the more likely we are to see the problems related to completeness and missing requirements.

Operating concept and use-cases. One of the most important techniques for identifying stakeholders and their associated requirements is to follow the behavior of a system within its intended environment, interacting with users and others (Hooks and Farry 2001). Historically this has been referred to as the "operating concept" (ANSI/AIAA-G-043-1992). Aspects of this have become formalized in object-oriented analysis with the label of "use-cases". We can use a use-case template to identify the basic information required for a given function or use-case (Figure 4). The collection of such use cases will constitute the set of behaviors required for a given system. Yes this still begs the question of how to establish completeness of such a *set* of use cases or the associated operating concepts. Functional analysis covers some of the same topics as use cases, but in a typically less disciplined manner. As such, it is very hard to know what might be missing. The more formal use-case approach has been adopted in both software and systems engineering to help ensure more complete system definition, but still begs the

[2] Bunch of Guys and Gals Sitting Around a Table.

question of when the set of use-cases is actually complete. Design constraints are not addressed at all.

Review. Reviewing the requirements by one or more stakeholder groups has been a traditional method for ensuring completeness. This is especially important in the area of design-constraint requirements where there are few other approaches available.

Use Case Element	Values
Use case name	
Iteration	
Basic course of events	
Alternative paths	
Exception paths	
Extensions	
Trigger	
Assumptions	
Preconditions	
Post conditions	
Related business rules	
Author	
Date	

Figure 4. Use-case description template.

A Formal Approach to Completeness

Stakeholder identification and interface quantification. A formal approach to completeness was proposed by (Carson 1998) as a 3-step process. It relies on capturing the stakeholder context and quantifying all developmental, operational, and maintenance interfaces. This is captured in Figure 5.

A statement of requirements can be considered as a problem to be solved. The solution is the design which satisfies the requirements. Thus, identifying the problem is the first step in stating the requirements.

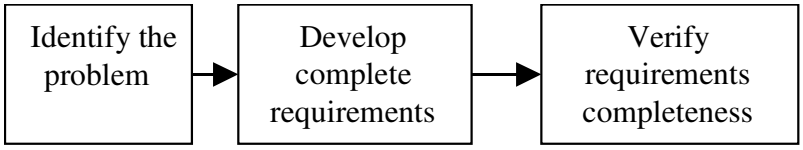

Figure 5. Requirements completeness process.

Developing the complete set of requirements is equivalent to completely stating the problem to be solved. As (Carson 1998) notes,

> Once the key interfaces are identified and quantified, the requirements associated with them can be defined and further analyzed to develop completeness. *The test for completeness of the problem statement step is that all stakeholder interfaces are identified and quantified for all applicable development, assembly, operations, maintenance, and disposal phases and related operating modes*.

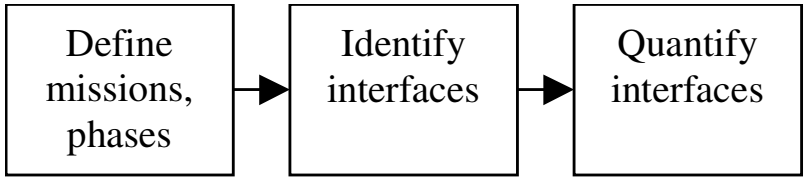

Figure 6. Steps in identifying the problem to be solved (after Carson 1998).

The problem identification phase can be broken into three parts: (1) identification of mission and associated life-cycle phases or operating modes, (2) identification of interfaces, and (3) quantification of interfaces (Figure 6). Identification of

interfaces begins with the context diagram (Figure 3), and includes associated functional and physical interfaces, and different stakeholders associated with development, production, operations, and maintenance (different context diagrams may be required for different phases).

Once the problem to be solved is clearly defined the next step is to define the required system behavior for each combination of contextual conditions, as in Figure 7.

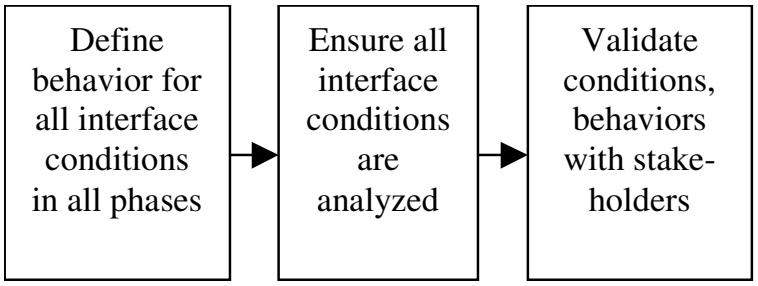

Figure 7. Once the interfaces have been defined, behaviors (requirements) must be defined for all combinations of conditions (after Carson 1998).

The next step to "define behavior for all interface conditions in all phases" can appear daunting at a first glance. Yet this step is the key to achieving completeness. For if we can demonstrate that we have identified a behavior (or design constraint) associated with all possible values of the interface conditions, then there are no remaining conditions possible (Carson 1995, 1996). The audit of *each and every* external interface condition for a companion requirement both determines the degree of completeness, and specifically identifies which requirements are missing. Similarly, auditing the stakeholder diagram (with the identification of the specific requirement or design constraint for which the stakeholder is the "champion") for each life-cycle phase determines the degree of completeness for these design-constraint requirements. The audit is with respect to the interfaces and design constraints identified and "owned" by various stakeholders or interacting systems; there is no absolutely correct answer, as we discussed in the first sections.

A simple example follows that may suffice; the reader is referred to the source papers for further discussion.

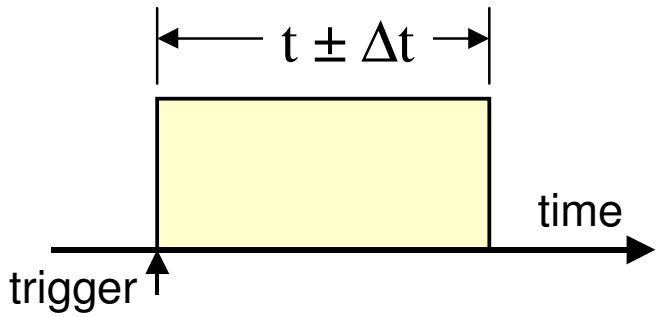

Figure 8. The time, t, over which the requirement is to be performed is measured from a specific trigger. Responses for all possible values of t must be defined (after Carson 2001).

Consider requirements associated with timing, where a change of state is desired a time t after some well-defined trigger condition Figure 8. Assuming that a tolerance is levied on t ($\pm \Delta t$), we must also define what should happen for the trigger condition appearing between t=0 and t-Δt, and between t+Δt and t=∞. For a given state of the system, this constitutes completeness in the antecedent ("trigger") condition. If a system behavior is specified for each of these three trigger conditions, then the system requirements as regards this interface (trigger condition) are, *complete*. If this process is repeated

for all interfaces and their associated parameters in all modes, states, and life-cycle phases, then the resulting set of requirements will be, *complete,* because there are no conditions associated with this parameter for which there is not a requirement identified.

It is obvious that combinations of conditions can approach an infinite set. To practically bound the problem the sets of requirements must be cast in terms of a smaller set of critical conditions, such that the values of other interface conditions are of no concern (*i.e.*, the required behavior is unaffected by the specific value of any other interface condition).

Stakeholder involvement is crucial in order to ensure that the derived behavior is consistent with expectations, especially for conditions that were not anticipated by one or more stakeholders (*e.g.*, specific failure conditions).

Similarly, design-constraint requirements are captured using the same process of stakeholder identification, parameter identification and quantification during all life-cycle phases, and completeness analysis. This can be likened to finding a "champion" for each requirement, one for whom deletion of same will bring about swift disapproval of the set of requirements. For example, a critical design constraint for Safety personnel might be that carrying and/or lifting handles or hook receptacles be in place (because of size or weight of the object). These are captured as an outgrowth of the operating concept and interface identification for manufacturing, test, installation, or maintenance activities. The quantification of the interface associated with lifting and carrying would be the size, shape, and location of such handles or hook receptacles, and identifying during which life-cycle phase they must be available.

Once the requirements have been developed, their completeness can be formally verified by determining (a) that all contextual conditions have been addressed in the requirements and (b) the documented responses are correct from the perspective of the stakeholder (Figure 9).

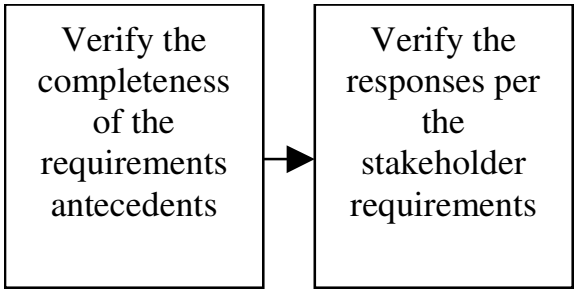

Figure 9. Verification of requirements completeness (after Carson 1998).

Summary

We have captured the processes and techniques to ensure that a set of requirements derived as part of a system development activity can be demonstrated to be "complete." Validation of these techniques, while part of the RWG mission, was not achieved within the reasonable schedule (given that this project began in 1997 or so). We look forward to any contributions to validating these processes.

Acknowledgments

The Requirements Completeness team of the RWG gratefully acknowledges the contributions to this work of those who began it several years ago. In particular, Pradip Kar, the RWG leader at the time, deserves special mention in this regard. We are also grateful to the members of the RWG for comments and insights provided during reviews and discussions.

References

- ANSI/AIAA G-043-1992, “Guide for the Preparation of Operational Concept Documents”, January 22, 1993.

Aslaksen, E.W., “Process for developing the Requirements Definition Document”, *The Changing Nature of Engineering*, McGraw-Hill, 1996, pp. 108-112.

Carson, R. S., “A Set Theory Model for Anomaly Handling in System Requirements Analysis,” *Proceedings of NCOSE 1995*, St. Louis, Missouri, USA.

Carson, R. S., “Designing for Failure: Anomaly Identification and Treatment in System Requirements Analysis,” *Proceedings of INCOSE 1996*, Boston, Massachusetts, USA.

Carson, R. S., “Requirements Completeness: A Deterministic Approach,” *Proceedings of INCOSE 1998,* Vancouver, BC, Canada.

Carson, R. S., “Keeping the Focus During Requirements Analysis,” *Proceedings of INCOSE 2001,* Melbourne, Victoria, Australia.

Carson, R. S., and Shell, A., “Requirements Completeness: Absolute or Relative?” ***Sys Eng***, Vol. 4, No. 3, 2001 (230-231).

- EIA-632, "Processes for Engineering a System," January 1999, page 67.

Gabb, Andrew, et al., “Requirements Categorization”, *Proceedings of INCOSE 2001,* Melbourne, Victoria, Australia.

Gonzales, R. “Completeness of Conceptual Models Developed using the Integrated System Conceptual Model (ISCM) Development Process,” *Proceedings of INCOSE 2001,* Melbourne, Victoria, Australia.

Grady, Jeffrey O., *System Requirements Analysis*, McGraw-Hill, New York, 1993, page 411.

Harwell, R. *et al.*, “What is a Requirement?”, *Proceedings of NCOSE 1993,* Washington, DC.

Hooks, Ivy, “Writing Good Requirements*,” Proceedings of NCOSE 1993,* Washington, DC, Volume 2.

Hooks, Ivy F., and Farry, Kristin A., ***Customer-Centered Products: Creating Successful Products Through Smart Requirements Management,*** American Management Association (New York, New York, 2001), p. 172.

- IEEE Standard 1220-1998, “IEEE Standard for Application and Management of the Systems Engineering Process”, 8 December 1998.
- IEEE 1233, “Guide for Developing System Requirements Specifications”, 1998.

Kar, Pradip and Bailey, Michelle, "Characteristics of Good Requirements", *Proceedings of INCOSE 1996, Volume II.*

Jones, D.A., “Executable Requirements Management Model, Interim Report - May 1998,” *Proceedings of INCOSE 1998,* Vancouver, BC, Canada.

Jones, D.A., Requirements Management Term Definitions, RWG posting 22.07.2002.

Mar, Brian W., "Requirements for Development of Software Requirements," *Proceedings of NCOSE 1994*, San Jose, California, USA.

- MIL-STD-490A, “Specification Practices”, 06/04/1985 (Cancelled 8/31/1995)
- MIL-STD-961E, (section 5.8) “Defense and Program-Unique Specifications Format and Content”, 8/01/2003.

Robertson, Suzanne, and Robertson, James, ***Mastering the Requirements Process***, Addison-Wesley (Harlow, England: 1999), page 5.

Schneider, R.E. & Buede D.M. 2000, “Properties of a High Quality Informal Requirements Document”, *Proceedings of INCOSE 2000,* Minneapolis, Minnesota, USA.

- *Systems Engineering Handbook,* Version 2.0, INCOSE (July 2000).

Wiegers, Karl E., ***Software Requirements***, Redmond: Microsoft Press (1999), p. 18.

Biographies

Dr. Ron Carson is an Associate Technical Fellow in Systems Engineering in the Integrated Defense Systems organization of The Boeing Company. During his career he has worked on requirements analysis for various military and commercial systems including locomotives, missiles, high-power lasers, the Boeing 777 Cabin Management System, and the Boeing Phased Array Communication Antenna System which evolved into the Connexion by Boeing[SM] service. He received "Best Paper" Award at the 1995 Symposium of the International Council on Systems Engineering for his work on "A Set Theory Model for Anomaly Handling in System Requirements Analysis". He is currently responsible for developing requirements for airborne intelligence, surveillance, and reconnaissance systems. He is a Senior Member of IEEE, an Adjunct Professor with the Department of Industrial Engineering Technology at Central Washington University, and holds two patents regarding methods of controlling satellite communications antennas. He has a Ph.D. in Nuclear Engineering from the University of Washington, and a BS in Applied Physics from the California Institute of Technology.

Dr. Erik W. Aslaksen is a Principal of Sinclair Knight Merz, the largest technology consulting firm in Australia. He obtained his M.Sc. in Electrical Engineering from the Swiss Federal Institute of Technology in 1962 and a Ph.D. in physics from Lehigh University in 1968. He is a Fellow of INCOSE and of Engineers Australia. Dr. Aslaksen's experience, gained in the US, Switzerland and Australia, covers fields as diverse as microwave components, power electronics, quantum electronics, and communications, and ranges from basic research to corporate management. In recent years his main interest has been in the area of systems engineering and engineering management, and before joining Sinclair Knight Merz in 1988, he was in charge of the design of the Australian Army's new tactical trunk communications system. He was the project manager for the EPCM of a major new, state-of-the-art underground copper mine, and has recently been involved in railway control systems, intelligent transport systems, and the business case for a new operational headquarters for the Australian Defence Force. He is the author of three textbooks (one with W.R. Belcher) and over fifty papers

Dr. Abd-El-Kader Sahraoui graduated with a BSC and MSC from Manchester University at UMIST (Faculty of Science and technology) in 1977 and 1979. He has held different positions in industry from 1979 to 1984 and obtained his PhD and DSc degrees from Toulouse University in 1987 and 1994. He is currently a Professor at Toulouse II University and researcher at LAAS-CNRS of the French research council. He has been an associate researcher at the University of California, Berkeley during the academic year 1999-2000. He is also co-founder of Veytek Techonologies.

Ron Kohl has been involved in the large systems development and integration business for 29 yrs. Ron was a part of the IBM team that developed and maintained the Space Shuttle Flight Software as well as supporting the Space Station (Freedom) Data Management System. Ron has also worked for the Lockheed Martin Federal Systems HQ Tech Staff and was the Chief Systems Engineer for the Titan Company's Civil Government Systems' NASA projects. Ron's areas of technical interest and activity include risk management, COTS-Based systems, process engineering/improvement, measurements, and system/enterprise architecting. Ron is a member of the AIAA, IEEE and INCOSE and is the former chair of the AIAA Software Systems

Technical Committee. He is also a member of the GEIA's G-47 (Systems Engineering) Technical Committee. Ron is now President of "R. J .Kohl and Associates", offering a full range of systems engineering and technical project management services.

G.F.J.Caple, MIEE, has been a Systems Engineer since 1955: Royal Air Force: 10years; Hawker Siddeley: 13years; GEC-Marconi: 20years. Martel/Buccaneer, SkyFlash/Phantom/ TornadoF3/Viggen, EW systems on many aircraft, creator of innovative integrated radar and EW systems, and bistatic AI. Creator and designer of the Generic Unified Systems Engineering Metamodel (GUSEM) Founder member INCOSE (UK). Initiator of GEC-Marconi Systems Engineering Process Group. Member of INCOSE RWG, IEWG, andAP233 Team. Member of IQA Integrated Management Special Interest Group. Lecturer and author with 11 papers published through INCOSE and DASC.

Dr. Regina Gonzales is currently a Systems Engineer in Product Realization Standards and Processes Department at Sandia National Laboratories. Her focus is on using modeling as a way to formalize requirements and is performing modeling of the Nuclear Weapons Complex Technical Practices. She is also the Requirements Lead on two technical teams developing requirements for major System Engineering projects. Dr. Gonzales is an Adjunct Faculty member at New Mexico State University. She is the Tech Board Communication Co-Chair, Co-Chair for the Requirements Working Group of the INCOSE, and the New Mexico Chapter President of emerging Enchantment Chapter. She has a Ph.D. in Computer Engineering with a specialty in Requirements Engineering from New Mexico State University, an M.S. in Computer Science from University of Colorado, an MS in Electrical and Computer Engineering from University of Arizona, a B.S. in Electrical and Computer Engineering from New Mexico University.

Paul Davies, MA, MIMA, C.Math, is a Technical Manager at the Leicester site of Thales Sensors, where he has worked since 1984. He graduated from Cambridge University in Mathematics in 1978 and has worked as a Systems Engineer at increasing levels of seniority continuously since then. He is Past President of the UK Chapter of the International Council on Systems Engineering (INCOSE), and co-chair of the INCOSE Systems Engineering Management Technical Committee. Paul has in the past held joint responsibility for the development of the Systems Engineering process at Thales Sensors, and is an active member of a team currently harmonizing prime contracting practices across Thales UK companies. For the last ten years, he has held positions as Design Authority and/or Engineering Manager on Electronic Warfare and Command Information System projects.