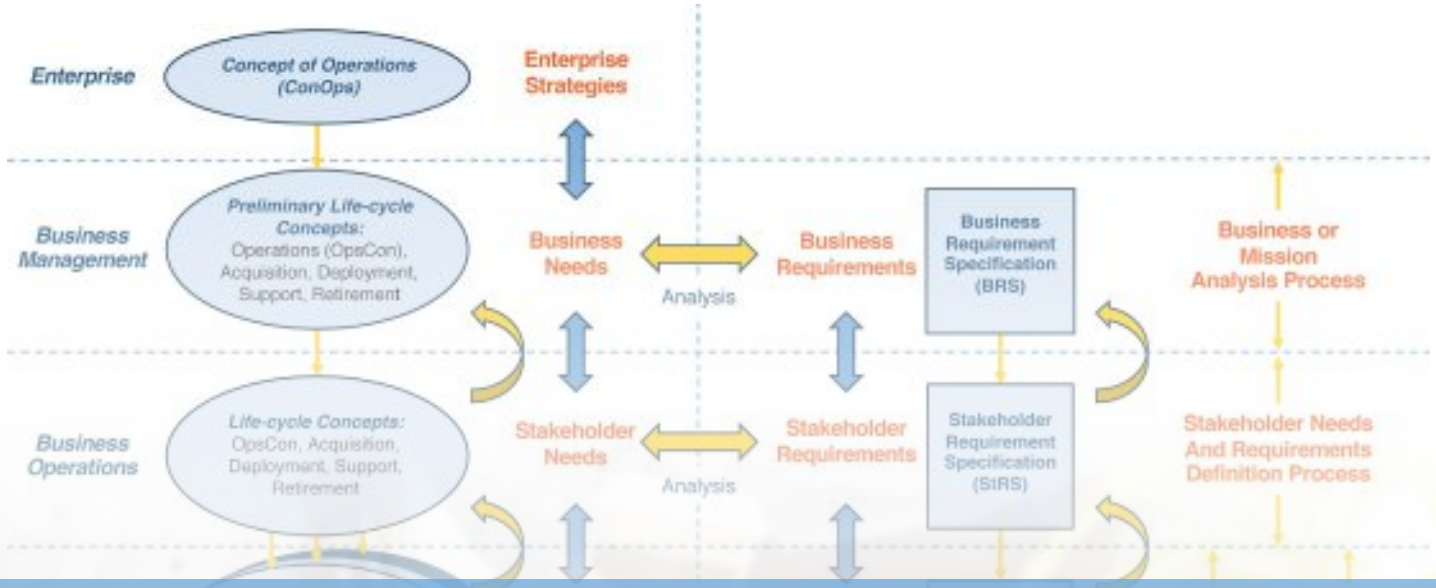
INCOSE



Guide for Writing Requirements

2019

Guide for Writing Requirements

Document No.: INCOSE-TP-2010-006-03
Version/Revision: 3
Date: 19 July 2019

Prepared by:
Requirements Working Group
International Council on Systems Engineering (INCOSE)
7670 Opportunity Road, Suite 220,
San Diego, California 92111-2222 USA

REVISION HISTORY

Revision	Revision Date	Change Description & Rationale
0	12 Jul 2009	Original
0.1	1 Jan 2011	Revision to use standard INCOSE formatting
0.2	30 Jan 2011	Revision to export from DOORS into standard INCOSE formatting
0.3	30 Jan 2012	Revision to accommodate comments from review by INCOSE Technical Operations
0.4	2 March 2012	Revision based on comments from INCOSE RWG leadership
1	17 April 2012	Revision based on comments from INCOSE RWG general membership
1.1	26 Jan 2013	Update at IW2013 by RWG to remove typographical errors
1.2	6 Apr 2013	Update to revise attributes and to amend rules after training packages developed
1.3	10 Feb 2015	Update at IW2015 by RWG at definitions, update characterizes and attributes.
1.4	10 Mar 2015	Updates resulting from the first round of reviews by attendees at INCOSE IW 2015
1.5	22 Mar 2015	Updates resulting from the reviews by RWG members
2	1 Jul 2015	Release by INCOSE Tech Ops
2.1	30 Jun 2017	Updates resulting from the reviews by RWG members at INCOSE IW2016 & IW2017
2.2	10 Feb 2019	Updates resulting from the reviews by RWG members at INCOSE IW2019
2.3	20 Mar 2019	Revision based on comments from INCOSE RWG general membership
3	19 Jul 2019	Release by INCOSE Tech Ops

Table of Contents

Preface

This document has been prepared and produced by a volunteer group of contributors within the Requirements Working Group (RWG) of the International Council on Systems Engineering (INCOSE). The original document was based on inputs from numerous INCOSE contributors, was edited by Jeremy Dick and published in draft form in December 2009 for internal INCOSE review. The original version was published as an INCOSE Technical Product in December 2009. Subsequently, the document has been revised extensively following reviews by the RWG led by Michael Ryan, Lou Wheatcraft, Kathy Baksa, and Rick Zinni at IW2013, IW2014, IW2015, IW2016, IW2017, and IW2019.

Authors

The principal authors of this Guide are:

Michael Ryan, *University of New South Wales, Canberra, Australia*
Lou Wheatcraft, *USA*
Rick Zinni, *Harris Corporation, USA*
Jeremy Dick, *Integrate Systems Engineering Ltd, UK*
Kathy Baksa, *Pratt & Whitney, USA*

Major Contributors

Those who made a significant contribution to the generation of this Guide are:

Ronald S. Carson, *Seattle Pacific University*
Jose L. Fernandez, *Madrid Technical University, Spain*
José Fuentes, *The Reuse Company, Spain*
Juan Llorens, *Universidad Carlos III de Madrid, Spain*
Robert A. Noel, *The Boeing Company*
Gina R. Smith, *Systems Engineering Global Inc, USA*
Christopher Unger, *GE Healthcare, USA*

Reviewers

The following reviewers submitted comments that were included in the update to this Guide:

Sascha Ackva, *Continental AG, Germany*
Luis Alonso, *The Reuse Company, Spain*
David Bar-On, *Motorola Solutions, USA*
Paul DeSantis, *GHSP*
Charles Dickerson, *Loughborough University, UK*
Jim van Gaasbeek, *Retired, USA*
Elena Gallego, *The Reuse Company, Spain*
Cecilia Haskins, *Norwegian University of Science and Technology, Norway*
Tim Kerby, *Edinburgh Systems Ltd*
Joel Knapp, *National Aeronautics and Space Administration, USA*
Ron Kohl, *R. J. Kohl & Assoc, USA*
Mario Kossmann, *Airbus, United Kingdom*
Avishek Kumar
Phyllis Larson, *Bechman Coulter, USA*
Robert Luff, *Loughborough University, UK*
Glenn Thibodeau, *Autoliv-Nissin Brake Systems*
Andreas Vollerthun, *KAIAO-Consulting, Germany*
Arie Wessels, *INCOSE South Africa*

Additional Copies/General Information

Copies of the *Guide for Writing Requirements*, as well as any other INCOSE document can be obtained from the INCOSE Central Office. General information on INCOSE, the Requirements Working Group, any other INCOSE working group, or membership may also be obtained from the INCOSE Central Office at:

International Council on Systems Engineering
7670 Opportunity Road, Suite 220
San Diego, California 92111-2222 | USA
E-mail: info@incose.org

Telephone: +1 858-541-1725
Toll Free Phone (US): 800-366-1164
Fax: +1 858-541-1728
Web Site: http://www.incose.org

1 INTRODUCTION

1.1 PURPOSE AND SCOPE

This guide is specifically about how to express concepts, needs and requirements in textual form in the context of systems engineering. The aim is to draw together advice from existing standards such as ISO/IEC/IEEE 29148 and from the authors and reviewers into a single, comprehensive set of characteristics, rules, and attributes for well-formed need and requirement statements.

This guide is not about the discovery, capture, or elicitation of requirements; nor is it about requirements analysis, requirements management and the development of models or design outputs, even though requirements play a key role in all of those activities and processes. The main focus of this guide is on how to express need statements (needs) and requirement statements (requirements) clearly and precisely as text, and in a form that supports further analysis and implementation, independent of any systems engineering (SE) tool used to capture and manage those needs and requirements throughout the system life cycle.

Although the focus of the guide is on writing textual needs and requirements, it does not, and should not, incorporate all the rules of good grammar and writing style. Rather, the guidance given herein assumes the writer has applied proper grammar and style—for example, applied proper punctuation, sentence structure, avoided double negatives, and appropriately located modifying clauses.

This guide addresses the characteristics that must be possessed by well-formed needs and the resulting requirements, as well as the subsequent sets of needs and sets of requirements. It provides a set of rules for writing needs and requirements that, if followed, will result in the relevant statements having the desired characteristics. The guide also includes a set of attributes that can be included with the need and requirement statements to form a complete need and requirement expressions (the distinction between a *statement* and an *expression* is discussed in Section 1.6). Finally, the guide also addresses the concept of a boilerplate, template, or pattern (see Dick and Llorens, 2012) for a requirement statement.

As systems become increasingly complex, the ability to share data and information, including requirements, across organizations both internal and external is critical to project success. Fundamental to forming shareable sets of data, the project must define and document a project ontology. The ontology includes the formal naming and definition of a set of terms, entities, data types, and properties as well as defining the relationships between these terms, entities, and data types that are fundamental to the project and organization of which the project is part. Having a documented project ontology helps ensure consistent use of this data and information across all system life cycle process activities and work products as well as across various groups within and external to the project. This common project ontology is key to interoperability and the ability to share sets of data and information, including needs and requirements.

In addition to the rules contained in this guide, authors of needs and requirements must also conform to the rules, processes and templates defined for their organizations. This guide can be used by organizations to develop and document a set of development rules, guidelines, processes, and patterns to be consistent with their unique culture, domain, and product line.

1.2 WHY USE THE TEXTUAL FORM OF COMMUNICATION?

Since English has many synonyms and words with slightly different shades of meaning based on context, the use of natural language to communicate needs and requirements can make it difficult to be clear, precise, and to avoid ambiguity. However, even though natural language can be an imperfect way of expression, textual forms of communication remain the only universal means of expression that covers the wide variety of concepts that must to be communicated throughout a system life cycle.

Of course, text is not the only medium by which needs and requirements can be expressed. Alternatives to writing textual statements for expression include:

- operational scenarios, use cases, and user stories (as used as part of Agile development methodologies, or epics, features and stories in the SAFe Framework);
- prototypes, such as used in production-driven and rapid application development methodologies;
- diagrams as part of a modeling approach with well-defined semantics, such as UML for software and SysML for generic systems; and
- tabular formats that provide template structures to collect and present requirements, such as Tom Gilb's *Planguage (*Gilb, T., 2005) or David Parnas´s SCR (Heitmeyer el al., 1997).

Just as there are issues with any form of technical communication, these other approaches can also be imperfect as they do not yet cover the wide range of concepts needed and have their own presentational, traceability, and management challenges.

- Problem statements, operational scenarios, use cases and user stories are written from the perspective of the user's (actor's) interaction with other actors and the system under development rather than the perspective of what the system under development must do in order for the users to interact with the System of Interest (SOI) in the way they expect, as defined by the use cases. While use cases are an excellent conceptual tool for stakeholder expectation analysis to help understand the features and associated functionality and performance expected by the stakeholders of the system of interest, they do not always effectively replace well-formed, text-based stakeholder needs and requirements for all the various ideas and concepts that must be communicated, especially non-functional needs and requirements.
- As an alternate form to communicate stakeholder needs and requirements, use cases, diagrams, and other model forms do not have the characteristics of well-formed statements as defined in this Guide that are necessary to communicate clearly the broad spectrum of needs and requirements into a language that can be clearly understood by all parties (stakeholders, developers, testers) over time.

The reality is that the textual form of needs and requirements are not only useful, they are necessary. Operational scenarios, use cases, diagrams, and other types of models are also useful and necessary. Each of these forms represents a specific visualization and need to be represented in an integrated data and information model of the SOI. With this approach, the underlying data and information model captures not only the needs and requirements but also their relationships to each other and to other artifacts represented within the data and information model. This enables stakeholders to view the data in whichever form is best for what they are trying to communicate or achieve. Therefore, it is important to understand which form and media is best to communicate specific ideas and concepts to a given audience over time.

For many ideas and concepts that need to be communicated; well-formed, text-based needs and requirements have been proven to be the most effective. Advantages include:

- ***Communication***. There is still (arguably, there will always be) a very sizeable audience who cannot interpret, do not understand, or who are not willing to work with, diagrammatic or other non-textual representations of needs or requirements, especially when the words (technical jargon) used are not intuitively obvious to the reader. These people, particularly the users of the system, may not have been trained in language-based models or find the terminology used in some diagrams and models to not be intuitive—in that case, effective communication has not taken place since the receiver of the message may not interpret and understand the message as intended by the sender and may well lose interest in the needs and requirements elicitation activities. On the other hand, text is universal. Of course, the text-based statements have to be well-written in such a way as to be clear, correct, and unambiguous; but then diagrams have even more potential to be unclear, incorrect, and ambiguous if they are poorly formed, defective, or the wrong meaning is assigned to them. Diagrammatic or model representation will almost always have to be supported by well written detailed textual statements in order for the representations to be understood unambiguously.
- ***Power of expression***. There is a wide variety of types of needs or requirements that must be expressed. Use cases, user stories, scenarios, diagrams, and models tend to focus on the functional architecture and may be capable of expressing functional, performance, and interface requirements but are not presently well suited to expressing non-functional requirements dealing with the physical architectural elements associated with quality (-ilities), regulations, standards, and physical characteristics. Textual forms carry the universal power of expression of natural language for all types of needs and requirements.
- ***Accessibility***. Even if stakeholders are willing to spend the time to learn modelling languages such as UML and SysML, the SE tools (including requirement management tools (RMT)) and modeling languages used to create and view the data and information represented by the model's dataset are not readily available and assessable to all stakeholders. In many cases, access is restricted by the number of "seats" or "licenses" purchased. Being able to provide needs and requirements in an electronic document format (pdf, or common office application formats) allows the stakeholders to view the needs and requirements in applications most will have installed on their workplace computers. In addition, there are still managers from the old school who still prefer, and demand, printed, text-based documents, and will continue to do so for the foreseeable future.
- ***Attributes***. Both the need and requirement expressions include attributes that can be used to manage them as well as the system under development across all life cycle process activities. While modeling languages allow users to define an entity having the name "attribute" and link that entity to a need or requirement, few practitioners do so. Operational scenarios, use cases, user stories, and other diagrams used to represent needs or requirements are generally not conducive to appending a set of attributes in the way they are defined in this guide.
- ***Formal, binding agreement***. Text-based requirement statements are more easily understood in a formal agreement or contract-based system development effort by a wider, and often, non-technical set of stakeholders including business management, project management, configuration management, contract administrators and legal support. Use of "shall" or another term defined to have the same meaning, makes it clear that what is being communicated is formal, the statement is a binding and will be verified. To be part of a binding agreement, especially in a legal contract, the requirements must be expressed formally and configuration managed in a form that 1) makes it clear the

statement is a binding and 2) has the characteristics of well-formed statements and sets of statements as defined in this Guide.

- ***System verification and validation***. Most formal agreement (contract) based product development and management processes include system verification and validation as formal processes that must occur prior to system acceptance and use. In highly regulated, safety critical industries such as the medical device industry, formal proof that the design outputs, including the product, meet the design inputs is needed prior to the product being able to be released into the market. Currently, no other form, other than textual requirements, has shown to meet these characteristics. (*See Section 1.7 for a more detailed discussion on the concepts of verification and validation*.)

Above all else, textual needs and requirements are a form of communication. As such it is vital that the intended message is clearly, and unambiguously communicated to those for whom the message is intended, over time. From a legal perspective, the responsibility of communicating the message is the responsibility of the sender, so the onus is on the writer for unambiguous communication of needs and requirements.

This guide refers solely to the expression of textual needs and requirements. If an organization chooses to use alternate forms to define needs and requirements, the characteristics defined in this guide are still applicable. If the alternate form does not have these characteristics, there is a risk that the needs and requirements of those for whom the system is being developed will not be met.

1.3 AUDIENCE

This guide is intended for those whose role is to *solicit*, *write, review,* and *manage* textual needs and requirements throughout the system development life cycle process activities, as well as those who *read, implement,* and *verify* that the developed system meets the requirements and *validate* that the developed system meets the needs in the operational environment. For the purposes of this guide, the focus is on design input requirements.

This guide is addressed to practitioners of all levels of experience. Someone new to this role should find specific guidance through the rules and associated examples provided herein to be useful, and those more experienced should be able to find new insights through the characteristics, rules, and attributes expressed, often absent from other texts, guides, or standards.

Although reading and reviewing needs and requirements requires less skill than writing them, it is helpful to understand why needs and requirements are expressed in the way they are. Understanding the characteristics of well-formed needs and sets of needs, requirements and sets of requirements, and the rules that need to be followed that result in having these characteristics will help both the authors as well as reviewers, developers, verifiers, and validators to identify defects, that if not corrected, can result in costly rework, schedule slips, and needs not being realized.

1.4 APPROACH

This guide presents underlying characteristics of need and requirement statements, practical rules for writing need and requirement statements, patterns that can be followed when writing need and requirement statements, along with attributes to be included with the need and requirement statements that result in a complete need or requirement expression.

All need statements, sets of needs, requirement statements, and requirement sets must have the characteristics expressed in this guide. The rules included herein provide guidance on how to achieve these characteristics. Focusing on characteristics is important because the set of rules alone can never be a perfect and complete expression of how to achieve the goal of well-formed needs and requirements. The diverse nature of needs and requirements means that the rules need to be adapted to particular situations. Given this reality, an understanding of the underlying characteristics informs the adaptation of the rules. Apart from this, human nature being what it is, many of us are better motivated to apply rules if we understand the reasoning behind them.

The attributes included in this guide also help to achieve the stated characteristics as well as aid in the formulation of the need and requirement expressions, system verification, system validation, management of the needs and requirements, and help to indicate applicability and enable reuse of the needs and requirements.

Pre-defined patterns for requirement statements provide a template or boilerplate for various types of requirements helping to ensure the requirement statements have the defined characteristics.

1.5 CONCEPTS

The characteristics and rules for writing needs and requirements, and the attributes that can be associated with those statements, are better understood if a few concepts are first explained.

When describing system development, some form of distinction is commonly made between *concepts*, *needs* and *requirements*. Concepts are typically narrative descriptions of ways in which the organization (and entities within an organization) expects to manage, acquire, develop, operate, support, and retire the business capability. Needs are formal statements of expectations for an entity stated in the language and perspective of stakeholders. Needs are transformed into requirements through a process of *requirements analysis* (which is also called *business analysis* or *mission analysis* at the higher levels)—there may be more than one requirement defined for any need. Requirements are formal textual statements that communicate what an entity must do to realize the intent of the needs. Both need statements and requirement statements are structured having the characteristics defined in this Guide. The SOI will be verified to have met the requirements that drove the design and validated to have met the needs from which the requirements were transformed.

As illustrated in Figure 1, concepts, needs, and requirements (see definitions in Section 1.6) exist at several levels (Ryan, 2013)—see also the “systems engineering sandwich” (Dick and Chard, 2004; Hull, Jackson, and Dick, 2011).

Figure 1 shows that there is an enterprise view in which enterprise leadership sets the enterprise strategies in the form of a Concept of Operations (ConOps) or Strategic Business Plan (SBP); a business management view in which business management derive business management needs and constraints as well as formalize their requirements; a business operations’ view in which business operations define the business operations needs and resulting requirements; and a system’s view in which the system to be developed is defined in terms of system needs and system requirements. Subsequently, at the lower level there are views of the system elements. Note that a system may comprise a number of elements including products, people, and processes—together these elements result in a needed capability for the organization. It should also be noted that Figure 1 best describes large

customer-contractor types of developments (such as in defense and aerospace contexts)—other models exist for market-led developments and for agile software developments.

At the highest level, the enterprise has several strategies that will guide its future. A system has its genesis in the enterprise ConOps or SBP which communicates the leadership's intentions regarding the management and operation of the organization—in terms of existing systems, systems to be developed or procured, services, and product line(s). At this level the ConOps, or SBP, defines the enterprise in terms of 'brand' and establishes a mission statement and corresponding goals and objectives, which clearly state the reason for the enterprise and its strategy for moving forward.

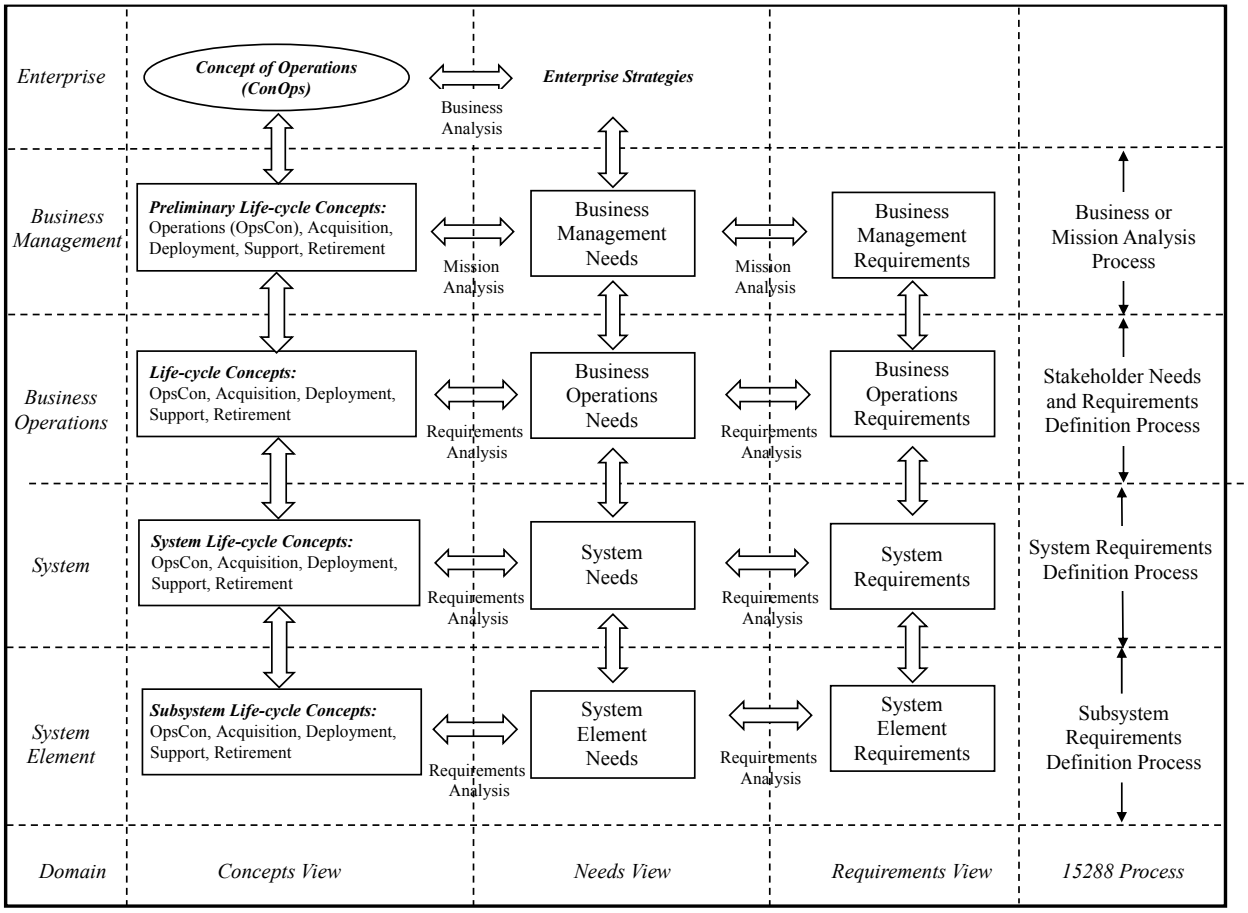


Figure 1. Transformation of concepts into needs into requirements (based on Ryan, 2013).

At the business management level, the Business or Mission Analysis Process begins with business management using the guidance in the enterprise ConOps or SBP to define the business needs, largely in the form of a set of high-level life-cycle concepts, which capture the business management stakeholder concepts for management, acquisition, development, quality, safety, security, human resources, marketing, operations, procurement, deployment, sales, support, and retirement. Chief among these concepts is the Operational Concept (OpsCon), which defines specific business needs at the business management level, which are then elaborated and formalized into business requirements, which are documented in the Business Requirements Specification (BRS), or Business Requirement Document (BRD), or Business Management Requirement Documents (BMRD). The process by which business management needs are extracted from the OpsCon and transformed into business management requirements is called *mission analysis* or *business analysis*.

At the business management level, requirements for governance of enterprise business management and operations are defined, as are the overall concepts to be used by business operations. Examples include business goals and objectives, principal capabilities and product line(s), branding, quality, safety, security, data governance, regulations and standards compliance, and high-level processes (for example, project management and systems engineering). At this level, these requirements can apply to the organization, program/project, and people developing a system as well as requirements on the systems (services and product lines) the enterprise procures, delivers, or develops.

Once business management stakeholders are satisfied that their needs and requirements are reasonably complete at the necessary level of abstraction, they are passed on to the business operations level for implementation. Here, the Stakeholder Needs and Requirements (SNR) Definition Process starts with the ConOps and the concepts contained in the life-cycle concepts as guidance. The requirements engineer (RE) or business analyst (BA) leads stakeholders from the business operations level through a structured process to elicit needs and develop life-cycle concepts, which are documented in the form of a business operations OpsCon or similar document (see Figure 1).

Specific business units, programs, and projects exist at the business operations level. At the business operations level, a key area of focus is on the infrastructure: programs/projects, people, processes, and tools used within the organization that will be involved in operations management and in supplying services or developing systems (products) for both internal and external customers.

More detailed concepts are defined to address the organizational capabilities needed by the business to conduct operations. These concepts include information technology (IT), data and information management, project management, systems engineering, product development, budgeting, scheduling, inventory control, configuration management, quality, safety, security, logistics, human resources, marketing, sales, etc. As defined above, mission analysis or business analysis processes are used to extract business operations needs from the business operations concepts and transform them into a formal set of business operations requirements.

Needs concerning the processes and tools are transformed into a formal set of stakeholder requirements, which are captured in what ISO/IEC/IEEE 15288 calls the Stakeholder Requirement Specification (StRS) but may also be called the Stakeholder Requirement Document (StRD), or Business Operations Requirement Document (BORDs). The form of the StRS/BORD includes Plans, Standard Operating Procedures (SOPs), Work Instructions (WIs), Process Definition Documents (PDDs), etc. A key point is that this set of requirements is on the organization, programs/projects, and people involved in business operations activities.

A second focus at the Business Operations level is on the concepts, needs and requirements concerning the services provided by the enterprise and systems (product lines) the enterprise develops or procures, to satisfy the needs of both internal as well as external stakeholders. At the business operations level, concepts for the services supplied and product lines supplied by the organization units are developed at a more detailed level. Business operations needs are extracted from these concepts and are transformed into a formal set of business operations requirements, which are also documented in the StRS. Again, that transformation is guided by a formal process of requirements analysis. A key point is that this second set of requirements is on the services supplied or systems being developed or procured, rather than on the organizational units supplying the service or developing or procuring the system or system elements as was addressed in the first set of business operations requirements.

At the system level, the focus is on a specific service being supplied or a specific SOI being developed or will be procured by a program or project within the organization for both internal and external customers.

The RE or BA uses the business operation's concepts, needs, and requirements as drivers or constraints on the specific service or SOI. The RE or BA uses a structured process to work with the various stakeholders, both internal and external, to understand the problem; define a mission or need statement, goals, and objectives; elicit stakeholder expectations; and identify risks, and drivers and constraints specific to the SOI. Structured processes used by the RE or BA to elicit specific stakeholder expectations include operational scenarios, user stories, storyboards, or use cases.

With this information the RE or BA works collaboratively with other organizational units (and customers) to define a feasible concept that will result in the stakeholder expectations to be met with acceptable risk and to be documented in some form of concept of operations (ConOps) or similar document. For further discussion of the Concept of Operations and the Operational Concept Document, and their interplay, see ANSI/AIAA G-043-2012e, Guide to the Preparation of Operational Concept Documents.

From this feasible system concept an integrated set of system level needs is defined and formally documented as the needs for the SOI. The RE or BA then transforms the set of needs into a set of requirements. While the focus of the business operations OpsCon is from the stakeholders' perspective, the system OpsCon is from the system perspective. At this level, the System Requirements Definition Process, the system needs in the system OpsCon and requirements in the StRS are then transformed by the RE or BA into System Requirements for the service to be supplied or the system to be developed, which are documented in the System Requirement Specification (SyRS) or System Requirement Document (SyRD) or equivalent electronic set of system requirements. The RE or BA transforms the needs into requirements using similar requirements analysis methods described above. For each need, the RE or BA asks: "What does the system need to do to fulfill the stated need?" The result will be one or more requirements on the service or SOI. The resulting requirements are documented, agreed-to, baselined, and placed under configuration management. Note that some organizations may prepare individual life-cycle concepts for each of the systems that are to be developed to meet the business needs. For systems that are outsourced or procured, the requirements on the organization supplying the service or developing the system are documented in a Statement of Work (SOW), while the system technical requirements are captured in the SyRS or SyRD type documents.

A set of requirements that has been documented, agreed-to, and baselined at one level will flow down to the next level as shown in Figure 1. At each level, the requirements are a result of the transformation process of the needs at that level as well a result of the elaboration (decomposition and/or derivation) of the requirements from the previous level. As such, SyRS or SyRD requirements are either decomposed from (that is, made explicit) or derived from (that is, implied by) the requirements of the system OpsCon and needs in the StRS or StRD or parent requirements in the previous level's SRD. Consequently, several systems, each described by the system OpsCon and SyRS, may be defined to meet the required capabilities. The same is true at the subsystem level, where the system element requirements may be either decomposed or derived from the subsystem needs and SyRS or SyRD.

In all cases, for each level shown in Figure 1, the set of requirements can be traced back to the needs from which they were transformed as well as the requirements at the previous level from which they were either decomposed or derived. The needs, in turn, can be traced back to their source—stakeholder concepts, needs, mission, goals, objectives, constraints, and risks.

Some companies may use a definition of "derived" to describe requirement to requirement relationships. In this case "derived" is defined as coming from a source other than a higher level requirement. In this case derived requirements do not have traces to higher level requirements. Therefore, the requirement should have associated with it rationale for why the requirement is necessary even though it does not have a parent requirement.

How requirements are expressed differs through these levels and, therefore, so do the rules for expressing them. As requirements are developed—and solutions designed—down through the levels of abstraction, we expect requirement statements to become more and more detailed. At the highest, governance level, the ideal requirement is a design input that is not specific to a particular solution and permits a range of possible implementations or solutions (design outputs). At the lowest level, requirement statements will be entirely specific to the selected solution.

It is important to note that the form of need statements and requirement statements at one level may not be appropriate for another level. For example, at the business management level, there may be a need or requirement that all products are “safe” or of the “highest quality” or project personnel develop products using “best practices” defined by organizations such as INCOSE, Project Management Institute (PMI), International Institute for Business Analyst (IIBA), or Software Engineering Institute (SEI). At the business management level, process requirements may be defined for implementing some capability level of Model Based Systems Engineering (MBSE) or Agile. While these are poor system level requirements, their form is appropriate for the Business Management level.

At the next level, business operations, there will be more detailed needs and requirements that define safe and quality in more detail. Project management and systems engineering offices will be formed that will be managed according to the best practices and processes defined at the business management level. These needs and requirements apply across all business operations activities, services, and product lines.

At the system level, concepts for achieving the desired level of safety and quality will be developed. Once a feasible concept to do so has been agreed to, specific needs concerning safety and quality will be developed and baselined. These needs will then be transformed in specific well-formed safety and quality requirements for that specific service or system having the characteristics defined in this Guide. These requirements will then be allocated to the system elements at the next lower level and the process repeats. The programs or projects responsible for providing the services or developing the systems will develop specific requirements for how they will implement the management and systems engineering process requirements defined at the business operations level.

It is important to differentiate between requirements on the systems to be developed from the requirements on the organization, people, and program/project responsible for developing those systems. A common issue is mixing the two types of requirements together rather than documenting them separately. Requirements on the developing organization belong in a project plan or SOW. Requirements on the service to be supplied or system to be develop belong in the set of technical requirements for the service or system.

Note: In the discussion concerning the model depicted in Figure 1, the term “level” was used. Sometimes, levels can also refer to layers. This is especially true when talking about levels of architecture. In the rest of this guide, the term “layer” or “level” can be assumed to mean the same thing. Caution should be used to make sure your intent is clear when using the following terms: level of organization, level of architecture, level of detail, high level requirements, low-

level requirements, requirement documents, and specifications. To address this ambiguity, some organizations have chosen to use the term "tier" to refer to levels of architecture.

The following is an example of why we must address levels of needs and requirements above the system level:

When an organization develops a system, it needs to use system thinking to address the macro system in which the system under development must work. This macro system includes organizational units above the project or program level that is developing a specific system, processes, and other systems with which the SOI must interact.

The enterprise, business management, and business operations levels are necessary because they represent three significantly different perspectives that must be defined and implemented in a top-down manner. Failure to do so is a common cause of failed projects.

Consider a wood-felling company whose management have just returned from a 1930s-trade show where they witnessed a demonstration of the revolutionary chainsaw that has been introduced to the market by Andreas Stihl's new company. Although the ability to cut down trees at a greater rate is very attractive, the company cannot simply nominate one of the stakeholders to sit down and write a system specification for the procurement of chainsaws. From a systems view, how will this new, transforming technology be phased into operations?

Further, management cannot even ask stakeholders at the business operations level to describe what they want from the introduction of the new chainsaw capability in the form of needs and requirements. The current operational managers are used to managing axe men who are not able (and probably not willing since they are going to be re-trained at best and, at worst, let go) to describe in any manner how the new tool is to be operated, since they have no familiarity with the operating procedures, training, safety, or the maintenance necessary for the new equipment.

Similarly, the logistics and procurement staff at the business operation level will know, in considerable detail, current logistic information such as the number of axe handles broken per linear meter of hardwood cut in support of current operations, but will know nothing of the support for a tool that will need a different maintenance methodology, significantly different support materials such as fuel and lubrication, different storage, and many more parts of much greater variety (such as chains, sparkplugs, and pistons).

Before the business operations level can even begin to address their requirements, therefore, the business management level above them must decide how the new capability is to be introduced into the organization. Is the reason to procure chainsaws to fell trees faster, or to cut down the same amount of wood more efficiently (with fewer operators, for example)? In either case, will the current tradesmen be retained and retrained, or will new operators be required? With axes, the tradesmen own their own axes and are responsible for maintaining the axes (keeping them sharp and replacing broken handles supplied by the company.) With the company procuring the chainsaws, what new logistics and procurement support will be required and how will it be acquired? How will the company transport materials—such as fuel and lubrication—that are not currently supported? How will the new capability be maintained (engine and chains), and by whom? How will the organizational structure need to be changed? How will the new capability be deployed—across all operators at once, or team by team, or region by region? If wood is to be produced faster, will additional transport vehicles be required to extract the product, and will additional sawmill capacity be required? Ultimately, of course, business management must also consider whether they want to produce more wood at the risk of flooding their own market resulting in a falling price and possibly reduced profits.

Before an acquisition can be considered, therefore, the business management level must draft the initial versions of the acquisition concept, operating concept, deployment concept, maintenance concept, and

retirement concept. Those concepts can then be fleshed out by the selected stakeholders from the business operations level before passing on to systems designers and procurement to develop the system specification and statement of work for the new chainsaws and supporting infrastructure and materials. In addition, procurement plans, maintenance manuals, operating instructions, safety plans, training, etc. can be developed.

Looking at the system development top-down then, the need for the three distinct levels above the system level is evident and we can see why the lack of business management and operations attention is the principal reason for project failure.

This same systems thinking thought process is needed when an organization wants to adopt any new technology or process. To successfully adopt new concepts like Agile, Lean, Six-Sigma, Model-based System Engineering (MBSE), etc., these three levels need to first be considered.

1.6 DEFINITIONS

Before considering the characteristics of well-formed need statements and requirement statements and the supporting rules for writing needs and requirements, we define several fundamental terms, based on the definitions provided in Ryan, Wheatcraft, Dick, and Zinni (2014). Many people may want to define needs and requirements for a service or system, but that need or requirement is not valid until it is formally agreed-to, baselined, and placed under configuration management.

Note: In these defintions we are using the term "customer" to refer to the organisation or person requesting a work product and to be the recipient of the work product when delivered. Customers are key stakeholders that exist at multiple levels and may be internal or external to the enterprise.

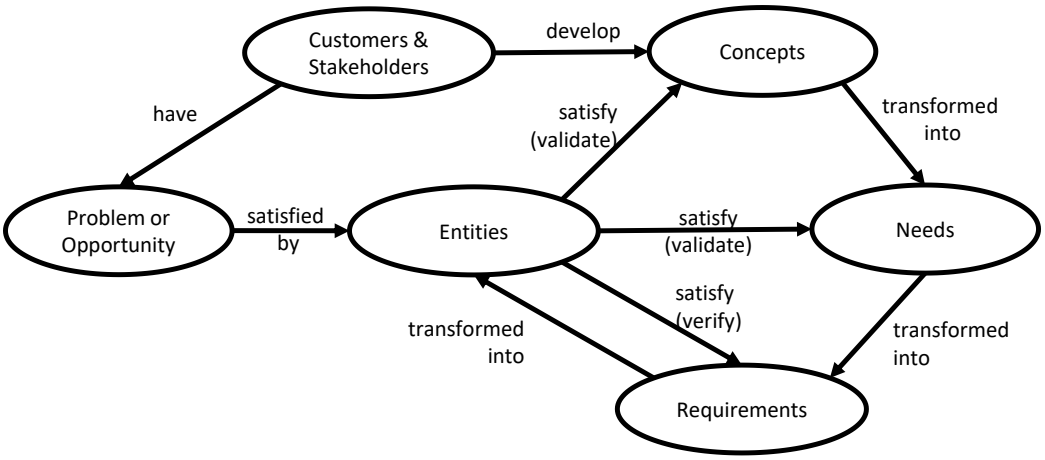


Figure 2. Entity-relationship diagram for customers, concepts, entities, needs and requirements terms (based on Ryan and Wheatcraft, 2017).

Needs and requirements apply to an entity, which could exist at any level of the model in Figure 1. Since terms such as "system" and "system element" are level-specific, we need a term that can apply at any level and to any single thing at that level. Each of these things is referred to as an "entity" which has needs and requirements that are to be met.

*An **entity** is a single thing to which a concept, need or requirement applies: an enterprise, business unit, service, system, or system element (which could be a product,*

process, human, or organization).

As shown in Figure 2, each of these things is referred to as an "entity" which satisfy concepts (validation), needs (validation) and requirements (verification) that are written for the entity as well as satisfy a problem or opportunity identified by the stakeholders. An entity is the subject of both a need statement ("The <stakeholders> need the <entity> to .....") and a subsequent requirement statement ("The <entity> shall .....").

Defining and documenting concepts, needs and requirements for an entity is more than just an exercise in writing; it is an SE activity where the RE or BA, through formal analysis, determines specifically what the customers need the entity to do to satisfy a specific problem or opportunity. This formal analysis starts with the RE or BA engaging the customers and other stakeholders in order to formalize a number of concepts which provide an implementation-free understanding of what is expected of the entity (design inputs) without addressing how (design outputs) to satisfy the mission, goals, and objectives to satisfy a problem or opportunity within defined constraints with acceptable risk. Concepts can be communicated is various forms including textual or graphical representations.

*A **concept** is a written or graphic representation that concisely expresses how an entity will satisfy the problem or opportunity it was defined to address* within specified constraints with acceptable risk.

There can be a number of concepts that apply to each entity, so it is useful to develop a minimal set of concepts that define the expectations of all entities. It is also useful to provide some structure to the set of concepts. As illustrated in Figure 1, life-cycle concepts include operations concepts, acquisition concepts, deployment concepts, support concepts, and retirement concepts. Concepts, needs and requirements are developed for entities at all levels.

Based on these concepts, the RE or BA then defines a set of needs, that when met, will result in the concepts being realized.

*A **need statement** is the result of a formal transformation of one or more concepts into an agreed-to expectation for an entity to perform some function or possess some quality (within specified constraints with acceptable risk).*

If a need statement results from a formal transformation of one or more concepts into an agreed-to obligation for an entity then, by decomposition, the key elements of the definition are: *formal transformation* and *agreed-to obligation*. Each of those two elements can be elaborated further by stating specific characteristics of a well-formed need statement (*These characteristics are defined in section 2.*)

- *Formal Transformation.* Including this term in the definition makes it clear that a need statement is the result of engineering analysis using one or more of the methods discussed earlier. Given the need statement is a result of a formal transformation, the following characteristics of a well-formed need statement have been derived:
    - *Necessary*
    - *Singular*
    - *Conforming*
    - *Appropriate*
    - *Correct*
- *Agreed-to Obligation.* Including this aspect in the definition makes it clear that, before a need statement is valid, both the customer and provider must agree with the need statement. Since the need represents an agreed-to obligation, the following characteristics of a need statement have been derived.

- *Unambiguous*
- *Feasible*
- *Able to be Validated*

As with developing needs, developing requirements is an SE activity where the RE or BA, through formal analysis, determines specifically what the system must do to meet the needs using a formal transformation process involving either decomposition or derivation.

Requirements at one level are implemented at the next level of the system architecture via allocation. A child requirement is one that has been transformed (derived or decomposed) from an allocated parent requirement; the achievement of the complete set of children requirements will lead to the achievement of the parent requirement. At the system level the achievement of the system level requirements results in the needs being achieved from which the system level requirement was transformed.

> *A **requirement statement** is the result of a formal transformation of one or more needs or parent requirements into an agreed-to obligation for an entity to perform some function or possess some quality (within specified constraints with acceptable risk).*

If a requirement statement results from a formal transformation of one or more needs or parent requirements into an agreed-to obligation for an entity, then by decomposition the key elements of the definition are: *formal transformation* and *agreed-to obligation*. Each of those two elements can be elaborated further by stating specific characteristics of a well-formed requirement. *(Note: These characteristics are defined in Section 2.)*

- *Formal Transformation.* Including this term in the definition makes it clear that a requirement is the result of engineering analysis using one or more of the methods discussed earlier. For each "need" the RE or BA asks: what does the entity have to do or what characteristic must it possess in order for the need or parent requirement to be realized? Engineering analysis results in one or more requirements. Given the requirement is a result of a formal transformation, the following characteristics of a well-formed requirement have been derived:
  - *Necessary*
  - *Singular*
  - *Conforming*
  - *Appropriate*
  - *Correct*
- *Agreed-to Obligation.* Including this aspect in the definition makes it clear that, before a requirement is valid, both the customer and provider must agree with the requirement statement. Since the requirement is to be a part of a fair agreement to meet an obligation, the following characteristics of a requirement have been derived.
  - *Unambiguous*
  - *Complete*
  - *Feasible*
  - *Verifiable*
  - *Able to be validated*

Both stakeholder need and requirement statements are more than the well-formed textual statement which is written succinctly in a standard format having the characteristics defined in this guide. As illustrated in Figure 3, the full need or requirement expression includes associated attributes that aid in the development and management of a need, the set of needs, requirement, and the set of requirements.

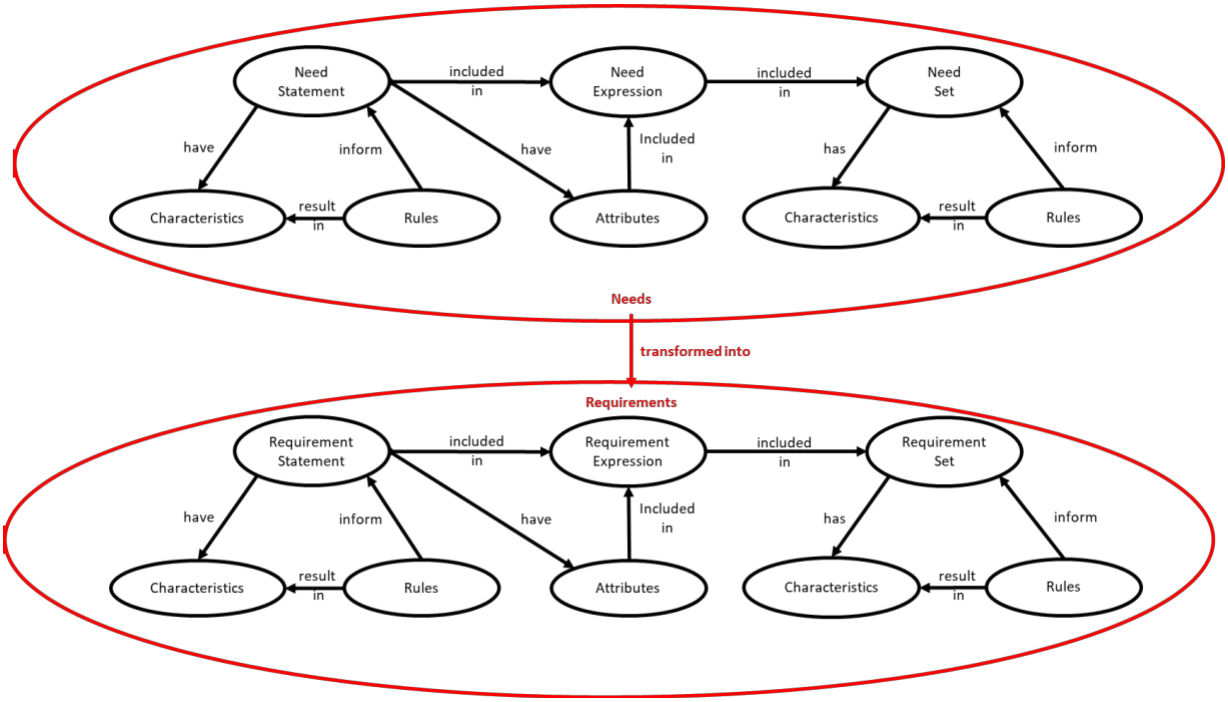


Figure 3. Entity-relationship diagram for needs and requirements terms (based on Ryan and Wheatcraft, 2017).

The following are definitions associated with need expressions and requirement expressions:

*A **need expression** includes a need statement plus a set of associated attributes.*

*A **requirement expression** includes a requirement statement and a set of associated attributes.*

*An **attribute** is additional information included with a need or requirement statement, which is used to aid in the management of that need or requirement.*

Attributes can be organized within four broad categories: (Note: Attributes are discussed and defined in more detail in Section 5).

- Attributes to help define a need or requirement and its intent.
- Attributes associated with the entity of interest and system verification.
- Attributes to help maintain a need or requirement.
- Attributes to show applicability and allow reuse of a need or requirement.

Although each individual need and requirement expression is important, it is ultimately the set of needs and resulting requirements that will describe the complete entity and it is the set of requirements that will be agreed-to as a contractual obligation. In some organizations the set of needs is baselined at a gate review prior to development of requirements. Some organizations may also contract with an organization stating only the set of needs in the contract and have the vendor develop a set of requirements based on those needs as part of the contract tasks.

*A **need set** is a structured set of agreed-to need expressions for the entity and its external interfaces captured in an Entity (Enterprise/Business Unit/System/System Element/Process) Needs Document or equivalent electronic representation of the set of needs.*

Given a need statement results from a formal transformation of one or more concepts into an agreed-to obligation for an entity, a set of needs results from the formal transformation of the set of concepts that represents an agreed-to obligation for the entity. Again, the key elements of the

definition are a *formal transformation* and an *agreed-to obligation*. Each of these two elements can therefore be elaborated further by stating specific characteristics of a well-written set of need expressions. (Note: These characteristics are defined in Section 3.)

- *Formal Transformation.* As the set of needs is the result of a formal transformation, the following characteristics of the need set have been derived:
  - *Complete*
  - *Consistent*

- *Agreed-to Obligation.* Since the set of needs represent an agreed-to obligation, the following characteristics of the set have been derived:
  - *Feasible*
  - *Comprehensible*
  - *Able to be validated*

Note that "feasible" is a characteristic of both the individual need statement and the set of needs. While each individual need may be feasible, the set of needs may not be feasible given the schedule and budget allocated to the development effort.

*A **requirement set** is a structured set of agreed-to requirement expressions for the entity and its external interfaces captured in an Entity (Enterprise/Business Unit/System/System Element/Process) Requirements Document or equivalent electronic representation of the set of requirements.*

Given a requirement statement results from a formal transformation of one or more needs into an agreed-to obligation for an entity, a set of requirements results from the formal transformation of the set of needs that represents an agreed-to obligation for the entity. Again, the key elements of the definition are a *formal transformation* and an *agreed-to obligation*. Each of these two elements can therefore be elaborated further by stating specific characteristics of a well-written set of requirements (Genova et al, 2013). (Note: These characteristics are defined in Section 3.)

- *Formal Transformation.* As the set of requirements is the result of a formal transformation, the following characteristics of the requirement set have been derived:
  - *Complete*
  - *Consistent*

- *Agreed-to Obligation.* Since the set of requirements is to be a result of a fair agreement to meet an obligation, the following characteristics of the set have been derived:
  - *Feasible*
  - *Comprehensible*
  - *Able to be validated*

Note that "feasible" is a characteristic of both the individual requirement statement and the set of requirements. While individual requirements may be feasible, the set of requirements may contain a quantity of requirements that is not feasible given the schedule and budget allocated to the development effort.

It should be noted that development of a new entity (greenfield SE), or an upgrade or modification of an existing entity (brownfield SE), can be performed as a result of various types of acquirer-supplier relationships. Development can be performed pursuant to a formal contract between two enterprises, or under a formal contract between two separate parts of an enterprise, or less formally between the end-user and the developer within a given element of an enterprise. Regardless of the formality of the acquirer-supplier relationship, formal documentation of the end-user and acquirer's requirements, as a rigorous transformation of the

needs, will lead to a smoother development project, resulting in a product fit-for-use and satisfying the customer expectations.

Note: The development of formal documentation of the end-user and acquirer's requirements is needed for groups that incorporate an "Agile" approach to development of their SOI. It is important that the boundary between the customer (person or organization requesting a work product) is clearly defined. This set of requirements should focus on "what" functionality, performance, quality, and standards are needed (design inputs), avoiding stating "how" (design outputs). This set of requirements includes requirements transformed from needs as well as interactions between the SOI being developed and other systems it interacts with (interface requirements). Within the Agile team, communication of internal requirements within the boundary may be less formal. As part of the Agile development activities, assuming the customers agree, the Agile team may involve the customer in the internal development activities. This involvement is beneficial when the true intent of a requirement is not clear or the customer may prefer one solution (how – design output) as compared to other solutions. However, once the SOI is ready for delivery, the team must perform system verification and system validation as defined in Section 1.7.

1.7 VERIFICATION AND VALIDATION

The terms "verification" and "validation" are used throughout this guide. While these terms are commonly used, the true meaning of the concepts represented in each are often misunderstood and the terms are often used interchangeably without making clear the context in which they are used resulting in ambiguity. To avoid this ambiguity, each term needs to be preceded by a modifier (i.e., the subject) which clearly denotes the proper context in which the term is being used, specifically need verification or validation; requirement verification or validation; design verification or validation; system verification or validation as shown in Figure 4. The concepts of verification and validation are very different depending on the modifier. When using these terms, it should be clear as to which concept is intended.

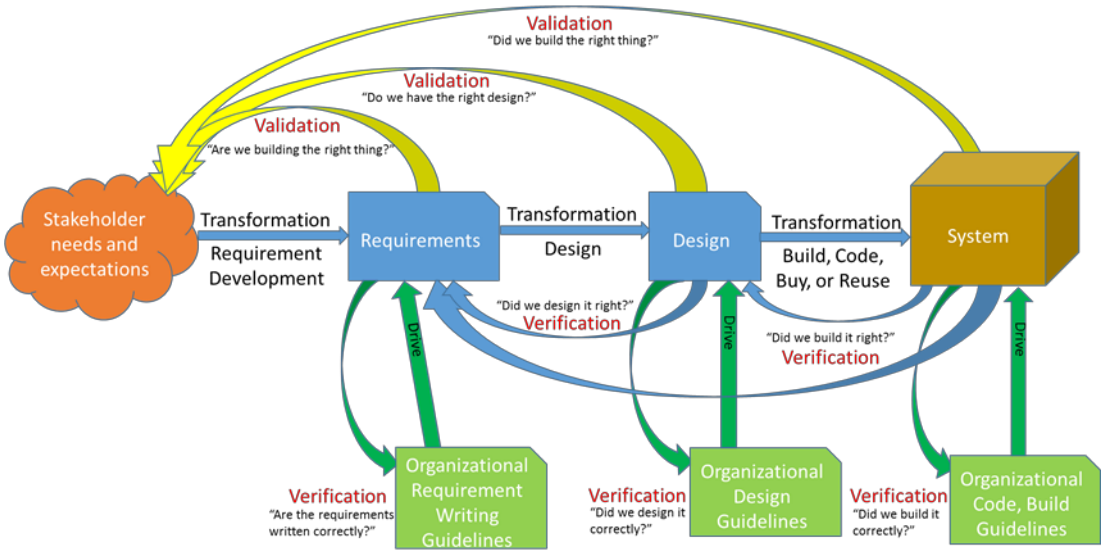


Figure 4. Verification and validation processeses confirm that systems engineering artifacts generated during the transformation processes are acceptable (Ryan and Wheatcraft, 2017).

Context example: As shown in Figure 4, a requirement set is the result of a **formal transformation** of needs and expectations contained within the set of needs.

Correspondingly, design is a result of **formal transformation** of the requirement set (design inputs) into an agreed-to design and corresponding build-to requirements/specifications (design output). The resulting system (design output) is the result of a **formal transformation** of the design and corresponding build-to requirements/specifications into a physical manifestation of the system.

The process of creating a requirement set involves:

- analysing problem or opportunity statements, stakeholder expectations, risks, and feasible concepts to obtain the necessary elements to be included in the set of needs;
- analysing needs to obtain the necessary elements to be included in the requirement set;
- selecting a format for the stakeholder need and requirement statements, selecting the attributes that will be included as part of the need and requirement expressions and organizing the need and requirement sets;
- identifying the characteristics of the stakeholder need and requirement sets against the organizational guidelines and rules by which the need and requirement statements and need and requirement sets are to be written; and
- transforming the needs into a set of requirements (design inputs) that unambiguously communicates these needs to the design organization.

In this context, *Requirement Verification* confirms, by inspection, that the requirements contain the necessary elements and possess the characteristics of a well-formed requirements, and that each requirement (and the requirement set) conforms to the rules set forth in the organization's requirement development and management guidelines. *Requirement Validation* confirms, by inspection and analysis, that the resulting requirement set meets the intent of the needs from which the requirements and the requirement set were transformed. Thus, both individual requirement statements and the requirement set are confirmed by both verification and validation.

To identify how we use the terms within this guide and to help remove the ambiguity in the use of the terms "verification" and "validation", the following definitions of these terms are included based on product life cycle context:

- ***Need or Requirement Verification***: the process of ensuring the need or requirement meets the rules and characteristics defined for writing well-formed need or requirement statements. The focus is on the quality (wording and structure) of the need or requirement statements. "Is the need or requirement statement worded or structured correctly in accordance with the organization's standards, guidelines, rules, and checklists?" These standards, guidelines, rules, and checklists would be developed at the business management and operations levels.

- ***Need Validation***: confirmation that the needs and set of needs clearly communicate the concepts from which they were transformed in a language understood by the requirement writers. The focus is on the message the needs and set of needs are communicating. "Do the needs and set of needs clearly and correctly communicate the agreed-to concepts, constraints, and stakeholder expectations?" or "Have we correctly and completely captured what the system needs to do?

- ***Requirement Validation***: confirmation that the requirements and requirement set is an agreed-to transformation that clearly communicates the needs in a language understood by the developers. The focus is on the message the requirements and set of requirements are communicating. "Do the requirements and requirements set clearly and correctly communicate the intent of the need set?" "Are we doing the right things?" or "Are we building the right thing *[as defined by the requirement set]*?"

Note: for this discussion the needs and requirements being referred to are design inputs. Also note that in many organizations' needs are not formally documented and configuration managed. This failure to do so is not consistent with the concept of "validation" as defined above. Sadly, the result is often a poorly formed set of requirements that do not have the characteristics defined in this Guide.

Need and requirement verification and *validation* activities should be done continuously as one develops the needs and requirements as part of the scope definition and requirement set baseline activities performed during the Scope or Concept Review and System Requirements Review (SRR) or similar type of gate review at each level.

Note: Some organizations do not make a distinction between requirement verification versus requirement validation. Rather they use only the phrase "requirement validation" to mean both. Using the phrase "requirement verification" often confuses the conversation in that many interpret "requirement verification" to have the same meaning as "system verification" as defined later.

Figure 4 also carries these concepts forward to design and realization of the system under development (design outputs).

Once the design input requirements set is baselined, the requirements are transformed into a design of the system (design outputs). Most organizations also have a dedicated set of "design principles or guidelines" or "golden rules" that guide the design process. Per the earlier discussion on levels of requirements above the system level, these design guidelines would be developed at the business management and business operations levels. These guidelines represent best practices and lessons learned the design team is expected to follow. As part of the design process, the design team may develop prototypes or engineering units. They will use these to run tests to fine-tune their design.

As part of the design process, there are usually a series of design reviews where the design outputs are both verified and validated (such as System Design Review (SDR), Preliminary Design Review (PDR), and Critical Design Review (CDR). In this context, *design verification* has two aspects: 1) Does the design clearly represent the design input requirement set that drove the design? and 2) Did the design team follow the organization's guidelines for design? Also, as part of the reviews, *design validation* is addressed to determine whether the resulting design for the system, when implemented, will result in the needs for the intended purpose/use being met in the operational environment.

Frequently, during the design reviews, the design team "pushes back" on design input needs and requirements that proved difficult to meet or were deemed not feasible for reasons of cost, schedule, and/or technology. This results in proposed changes which are submitted to the configuration control authority for the system for approval. When this happens, not only do the requirements need to be changed, but also the needs or parent requirements from which the requirements were transformed need to be examined, which could result in a scope change or changes to the parent requirements.

Design verification and *design validatio*n activities should be done as part of a continuous process during the design phase as well as during the baselining of the design in the final design review.

Based on this discussion, to help remove the ambiguity in the use of the terms "verification" and "validation", the following definitions for *design verification* and *design validation* are included in terms of a product life cycle context.

- ***Design Verification***: (1) The design verification process of ensuring design outputs meet the rules and characteristics defined for the organization's best practice guidelines for design. The focus is on the design process. "Did we follow our organization's guidelines for doing the design correctly?" As with the earlier discussion on levels of requirements above the system level, these standards, guidelines, rules, and checklists would be developed at the business management and operations levels. (2) The design verification process also includes ensuring both the design outputs are an agreed-to transformation of the design input "design-to" requirements into the design outputs and that the design is clearly communicated within the design output "build-to/code-to" specifications and that those requirements will result in a system that implements the design input requirements. "Does the design clearly and correctly represent the design input requirement set that drove the design?" "Does the design output "build-to/code-to" set of specifications clearly implement the agreed-to design?" "Did we design the thing right?" "Did we communicate the design accurately?"
- ***Design Validation***: Design validation is confirmation the design will result in a system that meets the needs for its intended purpose/use in its operational environment. "Will the design result in a system that will meet the needs that were defined during the scope definition phase?" The focus is on the message the design is communicating. "How well does the design meet the intent of the needs?" "Do we have the right design?" "Are we doing the right things?" "Will this design result in the needs being met?" Will the design output "build-to/code-to" set of specifications clearly communicate the agreed-to design to the builders/coders?" "Will implementing the agreed-to design output "build-to/code-to" specifications result in a system that will meet the needs in the operational enviroment?"

Once the design and resulting design output "build-to/code-to" requirements are baselined, they are transformed—via build, code, buy, or reuse—into the SOI. As with the discussion for the design process, most organizations have a set of "guidelines" or "golden rules" that guide the build (manufacture or code) process. These include workmanship and quality control requirements for the organization. Again, as with the earlier discussion on levels of requirements above the system level, these build-to guidelines are developed at the business operations level.

Once the SOI has been bought (procured), built, or coded, the concepts of *system verification* and *system validation* take on a more formal meaning. Thus, the SE life cycle processes include the design output processes of *System Verification* and *System Validation*. Each process represents a set of activities (test, demonstration, inspection, analysis) that cumulate with one or more gate reviews associated with the acceptance of the system by the customer.

After the system has been bought, built, or coded, there will be a gate review where it will be shown that the system was verified to have successfully met the design input requirements that drove the design. Thus, *System Verification* is a formal SE process and has a legal aspect where the developer is proving the baselined requirement set has been met. From a contracting perspective, the baselined design input requirement set is a type of contract which is legally binding.

In this context, *system verification* has three aspects: 1) "Does the built or coded SOI clearly represent the design input requirements that drove the design?" "Did we build the right thing?" 2) "Does the built or coded SOI clearly represent the design output "build-to/code-to" specifications?", and 3) "Did the build or code team follow the organizations guidelines for manufacturing and coding?'

Following *system verification*, *system validation* is performed. Again, *system validation* is a formal SE process and has a legal aspect where the developer is proving whether the bought, built, or coded and verified system, results in the needs concerning the intended purpose/use being met in the operational environment. Like the system requirements, the baselined design input needs defined during the scope definition phase can also be considered part of a contract and are legally binding.

Based on this discussion, to help remove the ambiguity in the use of the terms "verification" and "validation", the following definitions for *system verification* and *system validation* are included in terms of a product life cycle context.

- ***System Verification***: a process performed after design, build, procurement, or coding, making sure the designed and procured, built, or coded system meets its design input requirements. The focus is on the procured, built, or coded system and how well it meets the agreed-to requirement set that drove the design and fabrication. Internal to the organization, system verification activities are often done from a quality perspective to ensure the built or coded system meets its design output specifications. Methods used for *system verification* include: test, demonstration, inspection, simulation, and analysis. "Was the thing built or coded correctly?" Also, included in *system verification* is a determination that the organization responsible for building or coding the SOI followed the organization's rules, guidelines, and best practices associated with manufacturing and coding defined at the business management and operations levels. The focus is on the manufacturing or coding processes. "Did we follow our organization's guidelines for manufacturing or coding correctly?"
- ***System Validation***: a process that occurs after *system verification* that makes sure the designed, procured, coded, or built, and verified system meets the needs for its intended purpose/use in its operational environment. The focus is on the completed system and how well it meets the design input needs that were defined during the scope definition phase that should have occurred at the beginning of the project. "Did we build the right thing?" "Did we solve the problem correctly?"

System verification and *system validation* processes are directly related to the contractual obligation concept for a need and requirement statement and set of needs and requirements. It is through these process activities we prove we have met the design inputs - both the agreed-to requirements and the agreed-to needs of the entities who are the source of or own them. This is often accomplished as part of certification and acceptance activities. For medical and other safety critical systems, this process is required prior to a governmental organization authorizing the product to be released to and used by the public.

Note: When contracting for a system, it is important to make clear in the contract who is responsible for system verification and system validation and what the role of both customer and provider is during these processes. Is the contract complete upon successful system verification? Is the customer rather than the contractor responsible for system validation?

While the previous section presented definitions that are useful in helping address the issues of ambiguity in the use of the terms verification and validation, Systems engineering is more complex. Systems engineering is an iterative and recursive process. Requirements development and design occur top-down as shown on the left side of the SE "Vee" as shown in Figure 5.

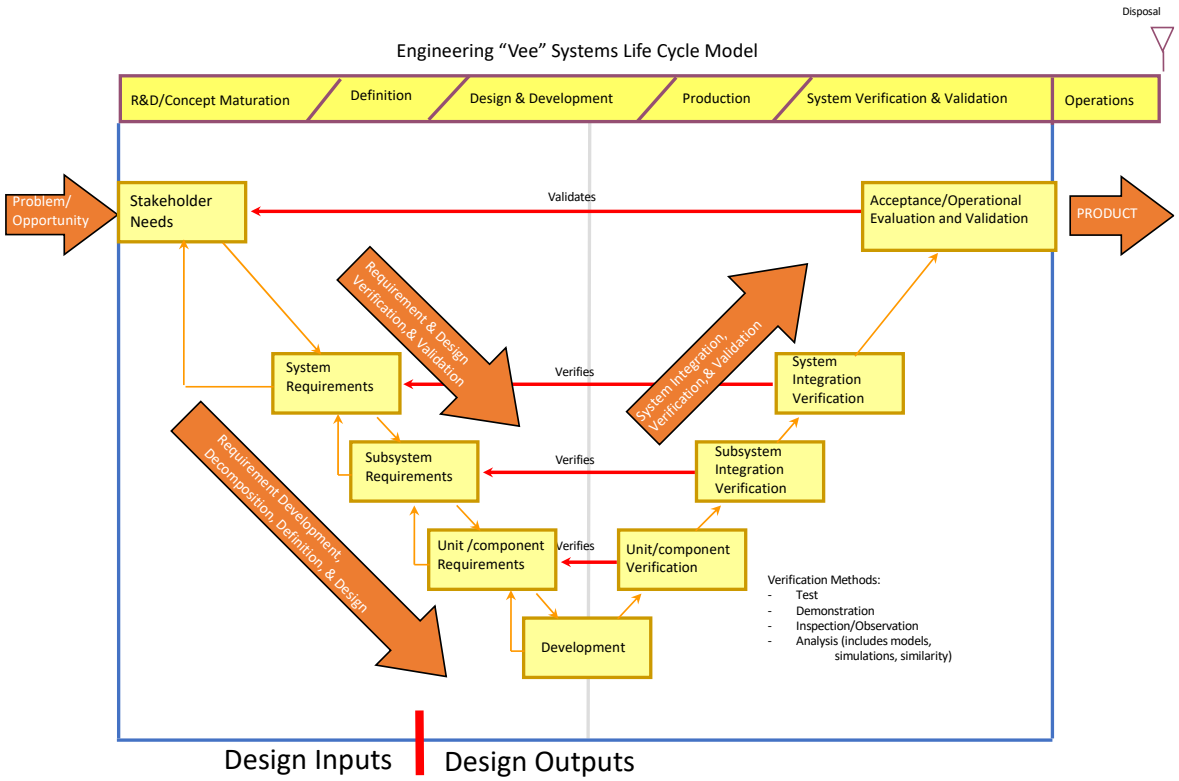


Figure 5: Verification and Validation and the Systems Engineering "Vee" Model (Ryan & Wheatcraft, 2017).

Systems engineering starts with the concept stage where the problem or opportunity to be addressed by the system is identified, stakeholder concepts are defined that will result in those expectations being meet with acceptable risk. Next, based on the feasible concepts, the set of needs are derived which are transformed into a set of system requirements that are baselined at a gate review via *requirements verification* and *requirements validation* as discussed above. Once the system requirements are baselined, design results in a system architecture in which the subsystems are defined. The design at this level is baselined via the *design verification* and *design validation* process discussed above.

For each subsystem, the above cycle is repeated with the definition of subsystem needs are defined and transformed into subsystem requirements, which are baselined via *requirements verification* and *requirements validation* as discussed above. Once the subsystem requirements are baselined, design results in a subsystem architecture in which the units/components are defined. This design is baselined via the *design verification* and *design validation* process discussed above. This process repeats until the organization makes a buy, build, code, or reuse decision. (*Some organizations may bring in an outside team to perform requirement and design verification and validation independently from the developing organization. When this is done, this is often referred to as Independent Verification and Validation (IV&V)*).

Note: In practice, especially when there is an integrated, collaborative, multidisciplinary team responsible for the system development, design takes place concurrently with scope definition and design input requirement development. This approach is preferred in that the design team is key in assessing concept feasibility. The challenge is being able to focus on what are design inputs versus design outputs. At some point, as the team moves down the left side of the SE "Vee" there is a transition between design inputs (design-to, what) and design outputs

(build/code-to, how). Some organizations consider this transition to occur between the conceptual or functional architecture and the physical architecture definition (design) phases.

Systems engineering integration*, system verification,* and *system validation* processes occur bottom-up as shown on the right side of the SE "Vee" as shown in Figure 5.

Once all the components that make up the subsystems are developed, *unit/component verification* and *unit/component validation* take place as described above. Once these activities are complete, the units/components are integrated and then the resulting subsystems are verified and validated as described above. Once the *subsystem verification* and *subsystem validation* activities are complete, the subsystems are integrated and then *system verification* activities are completed. In the end, proof will be documented that can be evaluated by the customer to determine *system verification* activities have been completed successfully showing that the design input requirements have been met (both organizational/people requirements documented in a project plan or SOW as well as the technical requirements).

Following *system verification* activities, *system validation* activities are performed. This could be done in the form of acceptance and/or operations evaluation and validation activities. Again, the customer organization must clearly define the customer/developer role in system validation activities. In the end, proof will be documented that can be evaluated by the customer (or governing body to determine *system validation* activities have been completed successfully, needs have been met, and the system will operate or support its specific use as intended in its operational environment.

Following the customer or governing body evaluation, the system can be accepted, and ownership transferred to the customer or released into the market. As stated previously, for medical and other safety critical systems, this process is required prior to a governmental organization authorizing the product to be released to and used by the public.

1.8 GUIDE ORGANIZATION

This guide focuses on the writing of needs and requirements and addresses the following aspects of needs and requirements: the *characteristics of individual need and requirement statements*, the *characteristics of sets of needs and requirements*, the *rules for individual need and requirement statements that help to formulate statements that have these characteristics*, the *rules for sets of needs and requirements that help to formulate sets that have these characteristics*, the *attributes of individual need and requirement statements,* and the patterns each well-formed need and requirement statement should follow.

This guide is organized as follows:

Section 2.0 defines the characteristics of individual need and requirement statements, provides rationale for the characteristics, and provides guidance for helping understand the characteristics.

Section 3.0 defines the characteristics of sets of needs and requirements, provides rationale for the characteristics, and provides guidance for helping understand the characteristics.

Section 4.0 defines the rules for individual need and requirement statements and sets of needs and requirements that help to formulate need and requirement statements and sets of needs and requirements. Included with each rule is an explanation of the rule and examples of the application of the rule.

Section 5.0 defines attributes that can be attached to a need or requirement statements to form need or requirement expressions. Also included is guidance on the use of attributes.

Appendix A lists acronyms and abbreviations used in this document.

Appendix B defines an overview the concept of patterns and lists examples of patterns that can be used for different types of requirement statements.

2 CHARACTERISTICS OF NEED AND REQUIREMENT STATEMENTS

This section defines the characteristics of individual need and requirement statements, provides rationale for the characteristics, and provides guidance for helping to understand the characteristics.

The following characteristics are currently in harmony with those listed in the SE Handbook, the SEBoK, and ISO/IEC/IEEE 29148 (for the first time) and only slightly different to those in ISO/IEC/IEEE 15288. The authors acknowledge that other sources may have slightly different lists of characteristics and that there are dependencies and apparent overlap between several of the characteristics (Carson 2018). The list of characteristics includes individual characteristics that provide an important perspective that is needed in order to craft well-formed requirement statements. Dependencies are addressed within either the rationale or guidance provided for each characteristic.

In defining needs and requirements, care should be exercised to ensure the need or requirement statement is appropriately crafted. The following characteristics of needs and requirements are elaborated in this guide.

2.1 C1 - NECESSARY

Definition:
The need or requirement statement defines an essential capability, characteristic, constraint, or quality factor needed to satisfy a concept, need or parent requirement. If it is not included in the set of needs and requirements, a deficiency in capability or characteristic will exist which cannot be fulfilled by implementing other needs or requirements in the set.

Rationale:
The formal transformation of a need from a concept must result in a need addressing a specific aspect of the concept that is necessary in order to meet the goals, objectives, stakeholder expectations, drivers, constraints, risks that are included in the concepts. Each need, individually or in combination with other needs in the set, must be sufficient to satisfy a specific concept from which it was derived. A need is considered sufficient if it, along with any siblings (common children of a single parent concept), satisfies its parent concept with acceptable margin (as determined by the program/project). For example, if the parent concept involves specific functional and performance expectations for a function, compliance with the expected functionality and performance described by the set of siblings' needs should ensure conformance to the parent concept by the integrated product. "Sufficient" both encompasses the characteristic "Correct" (C8) and enhances it to ensure it is not only an "accurate representation of the entity concept" but is sufficient to ensure the concept is satisfied by the child need. The transformation of a need into a requirement must result in a requirement that is necessary and sufficient in order to meet a need or set of needs for the entity from which it was transformed. Each requirement, individually or in combination with other requirements in the set, must be sufficient to satisfy a specific stakeholder need or parent requirement, including applicable conditions.

A requirement is considered sufficient if it, along with any siblings (common children of a single parent requirement), satisfies its parent requirement with acceptable margin (as determined by the program/project). For example, if the parent is a functional/performance requirement, compliance with the required functionality and performance level described by the set of siblings' requirements should ensure conformance to the parent requirement by the integrated product. Sufficient both encompasses the characteristic "Correct" (C8) and enhances it to ensure it is not only an "accurate representation of the entity need" but is sufficient to ensure the need is satisfied by the child requirement.

The development of children requirements via either decomposition or derivation must result in a set of children requirements that are necessary and sufficient in order to meet the intent of the need from which it is transformed or the allocated parent requirement the child is being developed in response to. Members of the set of children requirements can exist in different system element sets of requirements which the parent requirement was allocated to. In many cases these children requirements have a dependency, where a change in one could require a change in one or more of the others. It is important to link these dependent children requirements together. See also A31 – Trace to Peer Requirements.

Realization of every need and requirement requires resources, effort, and cost in the form of development, review, management, implementation, verification, validation, and maintenance. Unnecessary needs and requirements can lead to non-value-added work, additional cost, and unnecessary risk. Only necessary needs and the resulting requirements should therefore be included in the need and requirement sets. Once each need and requirement is proven to be necessary, the set of needs must then be a sufficient solution for the agreed-to concepts and requirements must then be a sufficient solution to the set of needs. Together, the sets of needs and requirements form the design inputs to which the design outputs will be verified and validated.

Guidance:

A need must be traced to a concept, driver or constraint, system need, or mission statement, goal, objective, risk, or stakeholder expectation defined during scope definition.

A requirement must be traced to a parent or source which could be one or more need(s) or higher-level allocated parent requirement(s).

Caution: When developing children requirements in response to a need or an allocated parent or source, avoid gold plating – the set of children requirement must be limited to only those that can be proven to be necessary and sufficient to meet the intent of the need, parent, or source to which they are traced.

There is no such thing as a "self-derived" requirement! All requirements must trace to a need, parent, or source (such as a design decision or trade-study output).

A need or requirement is not necessary (not needed in the set of needs or requirements) if:

- the need or requirement can be removed, and the remaining set will still result in the entity concept or needs being satisfied;
- the intent of the need or requirement will be met by the implementation of other needs or requirements;
- the need or requirement cannot be traced back to a source, need, or parent requirement; or
- the author cannot communicate the reason for the need or requirement (that is, it does not have a valid rationale).

One approach used by some organizations to limit the number of design input requirements is to take a "zero based" or "minimum viable product (MVP)" approach. Start by only including needs and requirements that are high priority and are critical or essential for an MVP. Then only add additional requirements that add value by reviewing each proposed need and requirement

against the mission, goals, and objectives as well as drivers, constraints, risks, concepts, and scenarios defined for the entity. If the need or requirement cannot be traced to one or more needs, parent requirement, or one of these sources, it is not necessary. The inclusion of rationale and other attributes defined in Section 5, such as trace to source or parent, for each need and requirement also aids in communicating the necessity and intent of the need or requirement.

For market focused product development, many companies will aim to elicit as much information as possible from stakeholders to help define the product being built. A process of evaluating requirements is used where they are considered based on their prevalence, urgency and the value realized by satisfying that need. Through the process of product management, a market can be defined consisting of stakeholders with similar product needs. At this point, a decision can be made on which requirements to include as necessary to satisfy the market needs.

Rules that help establish this characteristic:

R30 - /Uniqueness/ExpressOnce

Attributes that help establish this characteristic:

A1 - Rationale
A4 - Trace To Parent
A5 - Trace To Source
A31 - Trace to Peer Requirements

2.2 C2 - APPROPRIATE

Definition:

The specific intent and amount of detail of the need or requirement statement is appropriate to the level (the level of abstraction) of the entity to which it refers.

Rationale:

The wording of needs is often stated at a higher level of abstraction than is appropriate for a requirement. *See the example under R31, Abstraction/SolutionFree*.

Requirements may be imposed at any level; however, as a rule, a requirement should be expressed at the level of the entity to which it refers. Design input, design-to "what" requirements should state what needs to be stated for that level of the entity, not "how" the requirement should be met. However, design output, build-to/code-to "how" requirements purpose is to communicate to the builders/coders the agreed-to design, so they will reflect implementation. To avoid confusion, many organizations refer to design output requirements as "specifications" that often include drawings, logic diagrams, formulations, and algorithms in addition to textual requirements. ***For the purposes of this Guide, the focus is on design input requirements***.

A requirement stated at the wrong level is neither correct nor may not be verifiable at that level.

Guidance:

Needs are not requirements. They are stated from the perspective of the stakeholder's expectations for the SOI. Requirements are stated from the perspective of what the system needs to do to implement the needs. For example, a stakeholder need: "*The stakeholders need the system to comply with OSHA safety standards*". This states the stakeholder expectation for the system concerning safety. The requirements for the system transformed from this need statement would focus on the specific requirements the system must meet in order for the intent of this need

to be realized. Design output specifications would then communicate to the builders/coders of the design solution that meets the design input requirements.

Common mistakes made when documenting needs and the resulting requirements include:

1. Writing the needs as “shall” statements and calling these “stakeholder requirements” rather than needs. While the level of abstraction may be appropriate for a stakeholder need, it is not appropriate for a requirement. The result is often “stakeholder requirements” that don’t have the characteristics of well-formed requirements as defined in this guide.
2. Failing to formally document and manage the set of needs.
3. Including design output level requirements in the design input level of requirements.

Design input requirements must not be any more detailed or specific than is necessary for the level at which they are stated. In particular, the subject of the requirement needs to be appropriate to the level in which the requirement lives and the set of which it is a member. Unless there is a good reason, a requirement subject/noun should refer to the entity at the level of the requirement (not higher or lower).

The design input requirements avoid placing unnecessary constraints on the design at the given level. The goal for design input requirements is to be implementation-independent. There may be cases where there is good rationale for stating implementation. When this happens, the rationale must be included to make it clear why the specific implementation needs to be stated as a constraint to the design.

A useful question to ask of a requirement is “for what purpose?” or “why?” If the requirement is expressed in implementation terms (design output), the answer to this question may be the real requirement (design input).

A requirement should be stated at the level at which system verification and validation will be performed.

Lower level requirements stated at a higher level may seem like implementation. In this case, the real higher-level requirement may not be stated and thus not properly allocated to the next lower level. When this is the case, the requirement should be moved down to the appropriate level and the missing parent requirement added. Conversely, higher level requirements stated improperly at a lower level can be problematic because they may not be allocated properly to the other parts, resulting in missing requirements.

If the requirement is valid, but at a lower or higher level, determine what the appropriate level is (design input or design output) and document the requirement at that level.

It is good practice, once the next level of requirements is written, for the team to do a “leveling” exercise, where they look at each requirement at the higher level and determine whether or not it is at the appropriate level or should be moved down to a lower level or moved up to a higher level.

Rules that help establish this characteristic:
R3 - /Accuracy/SubjectVerb R31 - /Abstraction/SolutionFree

2.3 C3 - UNAMBIGUOUS

Definition:
Need statements must be written such that the stakeholder intent is clear. Requirement statements must be stated such that the requirement can be interpreted in only one way by all the intended readers.

Rationale:

A need or requirement statement must lend itself to a single interpretation of intent. An agreement is difficult to enact unless both parties are clear on the exact obligation. Ambiguity leads to multiple interpretations such that the stakeholder expectations may not be met.

The intent of a need or requirement must be understood in the same way by the writer, the designer, and those doing verification and validation activities following the "reasonable person" guideline. Ambiguity leads to interpretations of a need or requirement not intended by the author, and thence to ensuing problems, including project delay and even perhaps litigation and financial loss.

An ambiguous need is not correct nor able to be validated. An ambiguous requirement is not correct nor verifiable.

Guidance:

When writing a need or requirement statement, ask whether or not it could be interpreted more than one way. For needs, ask whether or not, it is able to be validated, i.e., whether it is stated in such a way that positive proof can be obtained that the stakeholder need has been met based on the wording of the need statement without having to interpret the stakeholder intent or make assumptions of that intent.

For a requirement ask whether or not the requirement is verifiable, i.e., whether it is stated in such a way that positive proof can be obtained that the requirement has been met based on the wording of the requirement without having to interpret the meaning or make assumptions as to the meaning.

The possibility of ambiguity is reduced by addressing these questions and applying the rules in this Guide.

Additionally, it is useful for the parties (stakeholders) who are involved in the implementation of the needs and resulting requirements or system verification and system validation to be involved in the development, review, and baseline of the needs and resulting requirements. When they see needs or requirements that are ambiguous and their intent not clear, they can identify the issue and suggest an alternate, unambiguous wording of the need or requirement statement. As a minimum, it is recommended that the need or requirement owner(s) take the development team and those involved in system verification and validation on a walkthrough of the need or requirement set to ensure that needs and requirements are understood, individually and as a set. As defined in Section 1.7, this activity is referred to as stakeholder need or requirement validation.

Due to the limitations of language, it may prove difficult to completely remove all ambiguity. In this case the use of the rationale attribute, A1, may provide additional insight of the intent to remove the ambiguity. The inclusion of contextual information with the requirement statement may help to reduce ambiguity. This may include supporting information or commentary on how the requirement was formed in the rationale.

When text only makes it difficult to communicate the intent of complex requirement, the inclusion of a diagram may help remove the ambiguity. See R23.

Rules that help establish this characteristic:

R1 - /Accuracy/SentenceStructure
R2 - /Accuracy/UseActiveVoice

R3 - /Accuracy/SubjectVerb
R4 - /Accuracy/UseDefinedTerms
R5 - /Accuracy/UseDefiniteArticles
R6 - /Accuracy/Units
R7 - /Accuracy/AvoidVagueTerms
R8 - /Accuracy/NoEscapeClauses
R9 - /Accuracy/NoOpenEnded
R10 - /Concision/SuperfluousInfinitives
R11 - /Concision/SeparateClauses
R12 - /NonAmbiguity/CorrectGrammar
R13 - /NonAmbiguity/CorrectSpelling
R14 - /NonAmbiguity/CorrectPunctuation
R15 - /NonAmbiguity/LogicalCondition
R16 - /NonAmbiguity/AvoidNot
R17 - /NonAmbiguity/Oblique
R18 - /Singularity/SingleSentence
R19 - /Singularity/AvoidCombinators
R22 - /Singularity/Enumeration
R23 - /Singularity/Context
R24 - /Completeness/AvoidPronouns
R28 - /Conditions/ExplicitLists
R32 - /Quantifiers/Universals
R33 - /Tolerance/ValueRange
R34 - /Quantification/Measurable
R35 - /Quantification/TemporalIndefinite
R36 - /UniformLanguage/UseConsistentTerms
R37 - /UniformLanguage/DefineAcronyms

Attributes that help establish this characteristic:

A1 - Rationale
A2 - SOI Primary Verification or Validation Method
A3 - SOI Verification or Validation Approach

2.4 C4 - COMPLETE

Definition:

The requirement statement sufficiently describes the necessary capability, characteristic, constraint, or quality factor to meet the need without needing other information to understand the requirement.

Rationale:

An agreement is not useful unless the obligation is complete and does not need further explanation. A well-formed requirement needs no further amplification to implement its intent. As an example, interface requirements should include a reference to the location of the agreement that defines how the entity needs to interact with the entity to which it interfaces (for example, an Interface Control Document (ICD) or Data Dictionary). Additionally, requirements based on a standard or regulation need to include a reference to the specific location within the standard or regulation from which it was derived.

Each requirement should be understood in its own right without the overhead of having to understand a number of other requirements.

Baselined requirement statements should not contain To Be Defined (TBD), To Be Specified (TBS), or To Be Resolved (TBR) clauses. TBx can be used during the analysis and definition process to indicate ongoing work but should not be in the final requirement set. Resolution of the TBx designation may be iterative, in which case there should be an acceptable timeframe for the TBx item to be resolved, determined by risks and dependencies. If the baselined set of requirement contains requirements with a TBx item, it must be made clear in supplier or vendor agreements (for example, SOWs) who is responsible to resolve these items and when.

An incomplete requirement also results in an incorrect and unverifiable requirement due to missing information (the requirement fails to address either "what", "how well", or "under what conditions".

Guidance:

To be complete, functional/performance requirements must have observable functions ("what"), measurable performance ("how well") and a statement of conditions ("under what conditions", for example, triggering events, environments, states and modes). Such requirements may be insufficient to satisfy the need or parent requirement because of these errors or omissions.

While fully appreciating that a requirement may require some context, the requirement statement itself should be a complete sentence that does not require reference to other statements to be understood in its basic form. Note, however, that a requirement can refer to other documents (for example, ICDs, standards, and regulations). When making these referrals, be specific to the sections of the documents that apply. Refer to a complete standard or regulation only if all the requirements within the standard or regulation apply to your specific system (which is a rare occurrence).

Caution: When referring to a standard or regulation, remember that standards and regulations are often written at a higher level of abstraction for a class of products similar to your SOI, but not necessarily your SOI specifically. It is a mistake to just copy and paste a requirement from a standard or regulation. Instead, you will need to derive requirements specific to your SOI at the appropriate level and then trace the resulting requirement(s) to the standard or regulation from which it was derived.

Requirements should not refer to one another through use of pronouns, nor should the understanding of the requirement assume the existence of a previous or subsequent requirement. This is especially important when requirements are managed within an application or database.

In a set of requirements contained in some form of "document", each requirement should be understood in its own right without having to understand the context as communicated in the requirements or headings surrounding it.

Finally, by following a specific pattern for a requirement statement, requirements tend to be "more complete", since these patterns provide guidance for specific information the requirement statement should include. See Appendix B for more information on patterns.

Rules that help establish this characteristic:

R6 - /Accuracy/Units
R7 - /Accuracy/AvoidVagueTerms
R8 - /Accuracy/NoEscapeClauses
R9 - /Accuracy/NoOpenEnded
R18 - /Singularity/SingleSentence
R24 - /Completeness/AvoidPronouns
R25 - /Completeness/UseOfHeadings

R33 - /Tolerance/ValueRange
R34 - /Quantification/Measurable
R35 - /Quantification/TemporalIndefinite
R39 - /UniformLanguage/StyleGuide

Attributes that help establish this characteristic:
A30 - Trace To Interface Definition

2.5 C5 - SINGULAR

Definition:
The stakeholder need or requirement statement should state a single capability, characteristic, constraint, or quality factor.

Rationale:
The formal transformation from a concept to a need can be a one-to-one or a one-to-many transformation, so both a source concept and resultant need statement(s) must each represent a single thought, aspect or expectation.

Similarly, the formal transformation from a need to a requirement can be a one-to-one or a one-to-many transformation, so both the source need statement and resultant requirement statement(s) must each represent a single thought, aspect or expectation.

The effectiveness of several processes associated with needs and requirements, such as decomposition, derivation, allocation, tracing, verification, and validation, depends on being able to identify singular statements. For instance, the verification information defined for a requirement can be far more precise when that requirement addresses a single capability, characteristic, constraint, or quality factor. A need or requirement with multiple thoughts is difficult to allocate and to trace to a parent or source.

A nonsingular need or requirement is neither correct nor verifiable.

Requirements patterns are useful ways to ensure singularity (see Appendix B).

Guidance:
Keep the need or requirement statement limited to one quality, characteristic, capability, function, or constraint. Understand how the statements fit into the allocation and traceability philosophy for the project.

Use the project standard patterns for writing needs and requirements.

Although a single need or requirement should consist of a single function, quality or constraint, it may have multiple conditions under which the requirement is to be met.

Avoid the use of the word “and”, R19, when it ties together multiple thoughts (phrases in the sentence), each of which may be allocated and verified differently. The presence of the conjunction “and” in a need or requirement statement should always prompt the writer to consider whether or not the statement is singular.

There are exceptions to the use of “and”. Some organizations allow the use of “and” to join two actions that will always be allocated, traced, and verified together. For example, there may be a requirement for a lit match to be extinguished and disposed into a container. Given that these two actions need to always be performed together (you would never do one action without the other), organizations may allow both actions to be communicated within a single requirement. In that case, however, it would be best to combine the two actions with a logical “AND”—see R15.

Rules that help establish this characteristic:
R9 - /Accuracy/NoOpenEnded R18 - /Singularity/SingleSentence R19 - /Singularity/AvoidCombinators R20 - /Singularity/AvoidPurpose R21 - /Singularity/AvoidParentheses R22 - /Singularity/Enumeration R23 - /Singularity/Context R39 - /UniformLanguage/StyleGuide

2.6 C6 - FEASIBLE

Definition:
The need or requirement can be realized within entity constraints (for example: cost, schedule, technical, legal, ethical, safety) with acceptable risk.

Rationale:
There is little point in agreeing to an obligation for a need or requirement that is not feasible. Agreeing to a need or requirement that cannot be realized with acceptable risk within constraints often results in project cost overruns and schedule slips. Inherently unachievable needs and requirements, such as 100% reliability, are at best a waste of time, and at worst lead to needlessly expensive solutions. An infeasible need or requirement cannot be satisfied because (a) it breaks the laws of physics, (b) it violates laws or regulations in an applicable jurisdiction, (c) it is in conflict with another requirement and cannot be concurrently satisfied, or (d) it leads to excessive program risk because of technical immaturity or inadequate margin with respect to program cost and schedule as a function of life cycle phase. An infeasible need or requirement is also not correct nor verifiable.

Guidance:
A need or requirement is considered feasible if, when considered along with other needs or requirements for a single system element (i.e., a set of entity needs or requirements), it does not cause an unacceptable cost, schedule or risk impact during the entity's life cycle. Feasible implies the existence of a possible feasible solution to satisfying a requirement. As shown in Figure 1 and stated in the definitions for needs and requirements, requirements are transformed from needs. Needs are derived from a concept for the SOI. With this in mind, feasibility starts with the concepts from which the needs are derived and the requirements that are transformed from the needs. If the organization has not defined a feasible concept, then the needs derived from that concept may not be feasible nor the requirements transformed from those needs. In general, it may be difficult to tell whether an individual need or requirement statement is feasible without an analysis of potential concepts for solving the problem or realizing the opportunity that is the driving reason for the development of the SOI. However, in some cases we can recognize and avoid needs and requirements that are clearly impossible or unrealistic (see R26).

Therefore, in most cases determining whether the need or requirement is feasible is difficult without an underlying assessment and analysis of the proposed concepts from which the needs are derived and requirements transformed. Before allowing a need into your set, an assessment of the feasibility of the chosen concept must be made in terms of the drivers and constraints including the maturity of critical technologies, cost, schedule, resources, regulations, higher level requirements, and risk. If not feasible within the stated constraints with acceptable risk, the need and resulting requirement should not be included in the set. Doing so can negatively impact cost and schedule and can result in a requirement that will not be met and verified. A good tool for assessing risk is the use of Technology Readiness Levels (TRLs) to determine and compute the maturity of a critical technology—a lower TRL represents more risk to the project than a higher TRL.

Other useful tools include modeling and prototyping to evaluate the feasibility of individual needs and requirements and subsets of needs and requirements. This means that the design team must be included in an integrated, multidiscipline, collaborative team responsible for the scope definition concept maturation activities needed to determine the feasibility of a concept in terms of physical implementation of the functional or conceptual model.

Feasibility may be determined for individual needs or requirements. The need or requirement is inherently infeasible where the requirement is either internally contradictory or it violates the laws of physics. More often, feasibility must be examined for sets of needs or requirements associated with a single entity to ensure there is no conflict in the set of requirements (next section).

The measurement of feasibility is not black and white. Measurement of feasibility is based on the degree of risk in successfully implementing the requirement within program constraints, unless precluded by physics (cannot be done, period) or requirements conflict. As stated above, TRLs can be a useful measure of risk for requirements dependent on the maturity of an essential technology. See also the attribute Risk (of implementation) A34.

Rules that help establish this characteristic:

R26 - /Realism/AvoidAbsolutes
R33 - /Tolerance/ValueRange

Attributes that help establish this characteristic:

A2 - SOI Primary Verification or Validation Method
A3 - SOI Verification or Validation Approach
A5 - Trace to Source
A34 - Risk (of implementation)
A35 - Key Driving Need or Requirement (KDN/KDR)

2.7 C7 - VERIFIABLE

Definition:

The requirement is structured and worded such that its realization can be proven (verified) to the customer's satisfaction at the level the requirement exists.

Rationale:

Unless a requirement is written in a way that allows design or system verification, there is no way to tell if it has been satisfied and that the obligation has been met.

Each requirement must have all the necessary information to be verifiable – there is clarity regarding what the requirement says, there is no ambiguity, and there are no missing characteristics within any requirement.

An unverifiable requirement can result in multiple, objective observers (for example, designers or testers) interpreting the requirement differently making it difficult to prove that the requirement was satisfied.

Verifiability is a necessary condition for establishing the characteristics: Appropriate (C2), Unambiguous (C3), Complete (C4), Singular (C5), Feasible (C6), Conforming (C9), Consistent (C11), and Comprehensible (C13). Therefore, verifiability should be addressed as the initial criterion and a basis for examining these other characteristics.

Guidance:

Write each requirement in a way that allows the design or system to be verified that the requirement has been met at the appropriate level by one of the four standard verification methods (inspection, analysis, demonstration, or test). Different kinds of requirements are verifiable in different ways, and this will influence the way the requirement is written. In general, to be verifiable, a requirement should be measurable.

A requirement is considered to be verifiable if:

1) a verification case can be determined from the requirement statement that allows for complete examination of all aspects of the requirement and precise determination of all values, including tolerances, such that success or failure can be determined, and
2) the requirement content is adequate to completely define the expected behavior, characteristics, conditions, and success criteria for the actual verification activity.

The measure of verifiability is the completeness and quality of the requirement: does it contain all the necessary information to establish what, how well, and under what conditions?

Each requirement must have all the necessary information to be verifiable – the wording of the requirement is unambiguous and there are no missing attributes. This can be facilitated by use of a standard template as described in Appendix B – see also characteristic Conforming (C9).

A verifiable requirement will have clarity and no ambiguity and will have the same meaning to all observers or readers.

A customer may specify, “The range shall be as long as possible.” This statement is ambiguous and unverifiable. This type of requirement is a signal that a trade study is needed to establish a verifiable maximum range requirement. *[This is a good example of a need written as a requirement. While “The stakeholders need the range to be as long as possible” is acceptable for a need statement, it is not acceptable as a requirement.]*

Each requirement should be verifiable by a single method at the level the requirement is written. If more than one verification method is required, it may indicate that the requirement should be broken into multiple requirements. It is possible to use one method to verify multiple requirements; however, such a possibility indicates a potential for consolidating requirements. Requirements need to be documented at the level at which they will be verified.

Note: It is important to not confuse method (test, demonstration, inspection, and analysis) with activities used as part of that method (testing, inspecting, analyzing). It is common for multiple verification methods to be listed, when there is a failure to make a distinction between method and activity when similar words are being used.

The most usual causes for a requirement not to be verifiable are:

- no clear definition of the correct functional behavior, conditions, and states.
- lack of accuracy or feasibility in the ranges of acceptable performance.
- use of ambiguous terms.

- failure to define a feasible concept and associated stakeholder need from which the requirement was transformed.
- the requirement is not feasible.

When writing requirements, use a verification point-of-view to imagine yourself performing the verification event (test, inspection, demonstration, or analysis,) and define what proof will result in the requirement's intent being achieved as a result of the verification event.

It is a recommended best practice to write verification requirements as part of writing the actual requirement. Verification requirements address the method of verification, the scope of activities involved, and the success criteria that will provide proof the system has met the intent of the requirement. When this is done, the quality of the requirement statement will improve resulting in this characteristic (verifiable).

Useful questions to ask of a requirement are:

- *How will I know if the requirement has been met?* If the requirement has been properly quantified, it will provide a precise answer to this question.
- *What are the mandatory and desirable levels of performance required?* The result of this may be that several values are provided describing the tolerance and trade-space allowed for this requirement.
- *Is what the requirement states what I want to verify?* If not, then rewrite the requirement to state what is intended.

For example, it is best to verify you obtain specific performance data rather than verifying the system has sensors to provide that data. It may have the sensors, but do they result in the actual quantity and quality of data needed? This is also a good example of design input versus design output. The need for the data is a design input, the sensors used to obtain this data are design outputs. All too often, when a design output requirement is stated there was a failure to state why the design output was needed – the proper design input. Because of this it could seem the system passed verification, but in reality, it didn't –the wrong requirement was verified!

Rules that help establish this characteristic:

R1 - /Accuracy/SentenceStructure
R2 - /Accuracy/UseActiveVoice
R3 - /Accuracy/SubjectVerb
R4 - /Accuracy/UseDefinedTerms
R5 - /Accuracy/UseDefiniteArticles
R6 - /Accuracy/Units
R7 - /Accuracy/AvoidVagueTerms
R8 - /Accuracy/NoEscapeClauses
R9 - /Accuracy/NoOpenEnded
R10 - /Concision/SuperfluousInfinitives
R15 - /NonAmbiguity/AvoidNot
R17 - /NonAmbiguity/Oblique
R18 - /Singularity/SingleSentence
R24 - /Completeness/AvoidPronouns
R26 - /Realism/AvoidAbsolutes
R28 - /Conditions/ExplicitLists
R32 - /Quantifiers/Universals
R33 - /Tolerance/ValueRange
R34 - /Quantification/Measurable
R35 - /Quantification/TemporalIndefinite

Attributes that help establish this characteristic:

A2 - SOI Primary Verification or Validation Method
A3 - SOI Verification or Validation Approach
A30 - Trace To Interface Definition

2.8 C8 - CORRECT

Definition:

The need must be an accurate representation of the concept from which it was transformed. A requirement must be an accurate representation of the need from which it was transformed.

Rationale:

Needs must be traceable to a source: concept, stakeholder expectation, drivers and constraints, goals, objectives, risks, etc. defined as part of the scope definition activities during the pre-concept and concept life cycle stages. The resulting needs transformed from these sources must be evaluated to determine the accuracy of the transformation.

A requirement must be able to be validated to ensure that the requirement communicates the right thing (for example, value, tolerance, and units) such that the need(s) (from which the requirement was transformed) will be met. It must be able to be shown that achievement of the requirement, as written, will result in meeting the intent of the need(s) from which it was transformed.

Correct implies “no errors” both from the perspective of the inclusion of incorrect information, the omission of required information, and avoidance of ambiguous wording.

A need or requirement cannot be correct if it is incomplete, ambiguous, unverifiable, inconsistent, unnecessary, or not feasible.

An incorrect need or requirement can result in a need that does not reflect the concept from which it was derived or a requirement that is not compliance with a stakeholder need or parent requirement from which it was derived.

Guidance:

Ensure the need statement reflects:

- an accurate interpretation of the concept, stakeholder expectation, drivers and constraints, goals, objectives, risks, etc. from which it was transformed;
- an accurate understanding of the underlying goals and objectives;
- the model or diagram from which the need was extracted so the need traces to the model; and
- the accurate representation of the underlying analysis and assumptions that were part of the transformation.

Use a defined development and management process to ensure accuracy of the transformation in the context of the individual need as well as the complete set of needs.

Ensure the requirement statement reflects:

- decomposition or derivation that is based on an accurate and unambiguous interpretation of the higher-level requirement or need;
- an accurate and unambiguous understanding of the underlying goals and objectives;
- the model or diagram from which the requirement was extracted so the requirement traces to the model or diagram; and

- the accurate and unambiguous representation of the underlying analysis and assumptions that were part of the transformation.

Incorrect information can mean having the wrong:

- values
- functions
- conditions, or
- other characteristics identified in the need or requirement.

Use a defined need and requirement validation process to ensure correctness of the transformation in the context of the individual need and requirement statements as well as the complete sets of needs and requirements.

Rules that help establish this characteristic:

R6 - /Accuracy/Units
R32 - /Quantifiers/Universals
R33 - /Tolerance/ValueRange
R36 - /UniformLanguage/UseConsistentTerms

Attributes that help establish this characteristic:

A2 - SOI Primary Verification or Validation Method
A3 - SOI Verification or Validation Approach
A6 - Condition of Use

2.9 C9 - CONFORMING

Definition:

The individual needs and requirements should conform to an approved standard pattern and style guide or standard for writing and managing needs and requirements.

Rationale:

When needs and requirements within the same organization have the same look and feel, each need and requirement statement is easier to write, understand, and review. Also, when conforming to an approved standard, the quality of the individual need and requirement statement will improve.

Conforming to standards or templates for needs and requirements will help to identify missing information resulting in the characteristics of complete, unambiguous, and verifiable.

Guidance:

For the derivation of a need from a concept to be formal, the structure of the resultant need statement must also be formal. For example, all needs may be required to be structured according to a specific pattern defined by the organization for the type of need statement: “The stakeholders need the <subject clause> to <action verb clause> <object clause>, <optional qualifying clause>.” If goals (non-mandatory) are included in the set of needs then the following pattern could be used: “The stakeholders would like the <subject clause> to <action verb clause> <object clause>, <optional qualifying clause>.”

For the transformation from a need to a requirement to be formal, the structure of the resultant requirement statement must also be formal. For example, all requirements may be required to be structured according to a specific pattern defined by the organization for the type of requirement statement: “When <condition clause>, the <subject clause> shall <action verb clause> <object clause>, <optional qualifying clause>.” See R1 for more detail on requirement styles and Appendix B for more detailed information on patterns.

Organizations must have well-defined standards and processes for needs and requirements development and management. These standards and processes must include rules for writing well-formed need and requirement statements, checklists to assess the quality of the needs and requirements, and templates for allowable structures for need and requirement statements. For example, all needs, and requirements may be required to have the characteristics and be written in accordance with the rules contained in this guide.

In many instances, there are applicable government, industry, and product standards, specifications, and interfaces with which compliance is required. When developing needs and requirements for a customer, the customer may have their own process, standards, checklists, and patterns. The people responsible for writing the needs and requirements may need to conform to the customer's processes and standards.

Rules that help establish this characteristic:
R12 - /NonAmbiguity/CorrectGrammar
R18 - /Singularity/SingleSentence
R30 - /Uniqueness/ExpressOnce
R36 - /UniformLanguage/UseConsistentTerms
R37 - /UniformLanguage/DefineAcronyms
R38 - /UniformLanguage/AvoidAbbreviations
R39 - /UniformLanguage/StyleGuide
R40 - /Modularity/RelatedRequirements

3 CHARACTERISTICS OF SETS OF NEEDS AND REQUIREMENTS

This section defines the characteristics of sets of needs and requirements, provides rationale for the characteristics, and provides guidance for helping understand the characteristics.

In addition to writing individual need and requirement statements that have the characteristics defined in the previous section, the following characteristics of a well-written set of needs and requirements must also be considered.

3.1 C10 - COMPLETE

Definition:
The need or requirement set for a given SOI stands alone such that it sufficiently describes the necessary capabilities, characteristics, constraints, interfaces, standards, regulations, and/or quality factors to meet the needs without requiring other sets of needs or requirements at the appropriate level of abstraction.

Rationale:
If the formal derivation of needs from an agreed-to concept results in individual needs that are necessary, the set of needs must be a sufficient representation of the concepts from which they were derived—that is, all necessary needs have been included to implement the concept, and that all unnecessary needs have been excluded. If the formal transformation of needs into requirements results in individual requirements that are necessary, the set of requirements must be a sufficient representation of the set of needs for the entity from which it was transformed — that is, all necessary requirements have been included to meet the needs, and that all unnecessary requirements have been excluded. Necessity and sufficiency can only be determined with respect to the set of specific stakeholder needs, the direct parent requirements that are derived and decomposed from the needs and

allocated to the system elements, or specific, derived anomalous conditions accepted by the stakeholders requiring a requirement.

Complete must also address the possible conditions a system may experience, including anomalies. Completeness therefore requires an additional examination of all possible conditions to ensure that the required behavior for specific conditions is consistent with stakeholder needs. These are the kinds of requirements ("what if?") that might not be addressed as part of initial analysis because the focus is on desired behavior under expected conditions. These additional requirements derived from anomalous conditions may not arise until the system development is underway and represent risk in system development and difficulty in ensuring completeness. Unspecified anomalous conditions can be handled using an "else" statement in the requirements. However, the required behavior still needs to be validated by the stakeholders

A set of needs or requirements that is not complete is also not correct. Missing information means there are needs or requirements missing (the set is incomplete and therefore incorrect). Missing needs or requirements can result in significant shortcomings in the delivered product pertaining to needed functionality/performance, robustness, quality, or conformance resulting in a failure to validate the SOI meets its intended use in the operational environment.

A set of needs or requirements cannot be correct if any individual need or requirement is not correct. The set of needs or requirements cannot be correct if it is either incomplete or contains any unnecessary needs or requirements.

Conversely, unnecessary needs or requirements inappropriately constrain the available solution space and cause extra program expense for developing, managing, implementing, verifying the unnecessary requirements, and validating the unnecessary needs. In the worst case, such unnecessary needs or requirements may over-constrain and compromise the overall system performance, leading to infeasible solutions which fail to satisfy necessary needs and resulting requirements.

Guidance:

The goal is to communicate clearly the needs for an SOI via the minimum set of requirements that are necessary and sufficient and no more. This applies no matter the level in the architecture the SOI exists.

For each SOI, a set of need statements represents a complete definition of the stakeholder expectations, concepts, drivers and constraints, risks to be mitigated, etc. both explicitly stated, and the implicit expectations not documented. Addressing implicit stakeholder expectations is a key part of defining the need set and deriving the resulting requirements that result in those needs being met, even if not originally stated.

A set of requirement statements represents a complete transformation of the needs for an SOI, both explicitly stated as needs and the implicit needs not documented (part of the derivation process). Addressing implicit needs is a key part of defining the requirement set and deriving requirements that result in those needs being met, even if not originally stated as needs. *Note: This shouldn't be an issue if the organization defined a complete set of needs as described in the previous paragraph. However, if this set of needs was not defined, then implicit needs could be an issue and will need to be considered when writing the set of requirements.*

A set of requirement statements also represent a complete transformation of higher-level requirements that were allocated to the SOI into a necessary and sufficient set of children requirements, that when implemented, will result in the intent of the allocated parent requirements being met. It is the responsibility of owners of the allocated requirements to ensure these children requirements exist and the responsibility of the owners of the receiving SOI to 1) validate the allocations were correct, 2) develop the set of children requirements, and 3) notify the higher-level systems of interest whenever a parent requirement is missing.

As an example, a set of software requirements may not be complete because not all relevant child requirements may be considered by the software architect if hardware limitations (for example, bandwidth, reliability, latency) are ignored. The hardware requirements and limitations may be missing or inconsistent with the assumed software architecture and software requirements. Therefore, it is important to consider the macro system of which the software is a part and include hardware stakeholders that can be the source of these missing needs and requirements.

Stakeholders are a primary source of requirements; leaving out a relevant stakeholder could result in missing or incorrect needs and the resulting requirements, resulting in expensive and time-consuming rework.

With today's increasingly complex systems it is almost impossible for a person or even a group of people to completely understand and manage every aspect of an SOI. A key tool to help define and manage these complex systems is the use of diagrams and modeling to ensure everything that is needed is included and what is not needed is excluded.

In the case of interfaces, the requirements need to refer to where the interaction between the SOI and other system is defined. In the case of standards and regulations, reference needs to be made to the specific requirements that apply to the SOI, rather than the whole document.

Completeness can be facilitated through the use of standard templates for need and requirement sets and thorough allocation and traceability.

Completeness of the set can also be facilitated through the use of the A38 – *Type/Category* attribute so that the set can be examined from a number of perspectives. Each organization will define types or categories in which a need or requirement fits, based on how they may wish to organize their requirements. The *Type/Category* field is most useful because it allows the database to be viewed by a large number of designers and stakeholders for a wide range of uses. For example, maintainers could review the database by asking to be shown all the maintenance needs and requirements, engineers developing specific test plans could extract all corrosion-control needs and requirements, and so on. See A38 for examples of use of types such as functional, non-functional, performance, and qualities.

Rules that help establish this characteristic:
R3 - /Accuracy/SubjectVerb R29 - /Uniqueness/Classify R41 - /Modularity/Structured

Attributes that help establish this characteristic:
A4 - Trace to Parent A7 - States and Modes A38 - Type/Category

3.2 C11 - CONSISTENT

Definition:
The set of needs contains individual needs that are unique, do not conflict with or overlap with other needs in the set, and the units and measurement systems they use are homogeneous. The language used within the set of needs is consistent (i.e., the same words are used throughout the set to mean the same thing).

The set of requirements contains individual requirements that are unique, do not conflict with or overlap with other requirements in the set, and the units and measurement systems they use are homogeneous. The language used within the set of requirements is consistent (i.e., the same words are used throughout the set to mean the same thing).

Rationale:

Conflicting needs and requirements lead to an incomplete solution space, and, if not identified early on in the development process, can lead to expensive rework.

For the transformation to be formal, the resultant set of individual needs and requirements must not conflict and must be consistent with each other.

Frequently, customer and other relevant stakeholder expectations or design constraints conflict with one another and need to be reconciled. Additionally, even if individual needs or requirements are unambiguous, the inconsistent use of terms, abbreviations, units, and measurement systems in different requirements results in ambiguity in the requirement set.

Needs and requirements that are inconsistent and conflicting with other requirements are not correct.

Needs and requirements that are inconsistent and conflicting with other requirements also result in a set of needs that are not able to be validated and a set of requirements that are not verifiable.

Consistency in needs and requirements wording is greatly assisted by the use of a centralized domain ontology that is shared among all stakeholders.

Guidance:

It is a challenge spotting conflicts between needs and requirements when the set of needs and requirements is large, as is often the case in today's increasingly complex, software centric systems.

It is important to identify needs and requirements that have relationships with other requirements, either directly or indirectly. This is especially an issue when one need or requirement is changed without making a corresponding change to the other dependent need(s) or requirement(s). One concept to help prevent this type of inconsistency is allocation and to link (trace) dependent requirements to each other.

When allocating functionality, performance or quality requirements to lower level entities of the architecture, in many cases the resulting children requirements are dependent. To ensure consistency throughout the development life cycle, these dependent requirements need to be linked together to ensure consistency is maintained when changes occur.

It can be difficult to identify conflicts merely through the language used to express individual need and requirement statements, but it can be made easier by classifying and grouping needs and requirements. One strategy is to classify them by the kinds of aspects they communicate (e.g. safety, timeliness, quality, functional area.) Then use sorting and filtering to bring otherwise diverse statements together and examine them for interactions, trade-offs, dependencies, and conflicts. Another strategy is to have the need or requirement statement with a common classification to follow the pattern defined for that classification.

Another key tool is the use of diagrams and other models that show the relationships between needs and between the resulting requirements. Using a software tool, e.g. diagraming and modeling, to manage dependencies can help in identifying conflicts and manage dependencies.

Define an ontology for the project and use a glossary or data dictionary to ensure consistent use of terms and abbreviations throughout the need and requirement sets.

Pay special attention to interface requirements to make sure they are consistent with the other systems your SOI interacts with. Ideally, both systems are managed within the same tool allowing interface requirements to be linked in applicable pairs, and traced to the document, e.g. ICD or data dictionary, that defines the interaction referenced within the interface requirements.

Rules that help establish this characteristic:

R4 - /Accuracy/UseDefinedTerms
R29 - /Uniqueness/Classify
R30 - /Uniqueness/ExpressOnce
R36 - /UniformLanguage/UseConsistentTerms
R37 - /UniformLanguage/DefineAcronyms
R38 - /UniformLanguage/AvoidAbbreviations
R39 - /UniformLanguage/StyleGuide
R40 - /Modularity/RelatedRequirements
R41 - /Modularity/Structured

Attributes that help establish this characteristic:

A30 - Trace To Interface Definition
A31 - Trace to Peer Requirements

3.3 C12 - FEASIBLE

Definition:

The sets of needs and requirements can be realized within entity constraints (for example, cost, schedule, technical) with acceptable risk.

Rationale:

Just as there is little point in agreeing to an obligation for an individual need or requirement that is not feasible, the sets of needs and requirements must be achievable within the appropriate constraints including cost and schedule. If infeasibility is not identified early in the development process, it can lead to wasted effort and cost.

Guidance:

While individual need and requirement statements may seem feasible, they may not be so when placed in combination with others. That is, the combination of feasible individual needs or requirements does not necessarily sum to a feasible set of needs or requirements. (Like the old saying “The straw that broke the camel’s back”.)

For example, the following are feasible, individual requirements for a laptop computer: weighs less than 1.4 kg, has a storage capacity of 1 TByte, has 4 GByte of RAM, has a wireless network interface, has an Ethernet network interface, has a USB interface, has an HDMI interface, can be dropped from 1m without damage, can survive in temperatures of ±50°C, and retails for less than $900. While each of those requirements seems perfectly feasible, even at first glance to a non-expert, we cannot be so readily sure that the set is feasible (that is, all requirements can be met simultaneously). As soon as we go past a couple of dimensions, our human intuition quickly deserts us.

If the related individual needs or requirements are distributed throughout the need or requirement set, it is even more difficult for us to “get our minds around” the feasibility of the set. As was

stated for individual need and requirement statements, determining feasibility of a set of needs or requirements is not always completely known and is often assessed in terms of acceptable risk consistent with the development life cycle stage.

In the solution space, there should be at least one and preferably multiple achievable concepts that will result in the problem being solved or opportunity to be realized that drove the need for the SOI. The concept should be technically achievable (such as in terms of technology maturity level and advancement), and achievable within the constraints of the project (such as cost, schedule, technical, ethical, and regulatory compliance) with a level of risk consistent with this life cycle stage.

Requirements within the same entity set of requirements are analyzed together with respect to program constraints and technology readiness to determine feasibility. Additional characteristics mapped to feasible include “Consistent” (C11) and “Able to be validated” (C14) for the set of requirements. An infeasible set of requirements cannot be correct, whether due to inconsistency, physical impossibility or excessive program risk. Feasibility of a specific requirement must be determined in the context of all requirements for a specific entity set and can be evaluated without regard to the parent requirements.

Before allowing a set of needs to be baselined, an assessment of the feasibility of the chosen concept needs to be made in terms of the drivers and constraints. If not feasible within the stated constraints, the set of needs will not be feasible nor will the resulting set of requirements that are transformed from that set of needs.

An approach that can be used is to consider the set of needs or requirements to be a “bucket” which is bound by cost, schedule, and technology. When adding individual needs or requirements to the bucket, whether or not the individual needs or requirements “fit” within these constraints must be determined. When the bucket is full, the addition of any more needs or requirements puts the project at increased risk. The bucket analogy is also useful for addressing change. If a bucket is full, and someone wants to add an additional need or requirement to the bucket, the question of feasibility must be addressed. Can the bucket be made bigger? Can something of lower priority be taken out? Is the project willing to accept more risk? If the answer is no, then the project must say no to the change!

Rules that help establish this characteristic:

R26 - /Realism/AvoidAbsolutes
R29 - /Uniqueness/Classify
R30 - /Uniqueness/ExpressOnce
R33 - /Tolerance/ValueRange
R34 - /Quantification/Measurable

Attributes that help establish this characteristic:

A24 - Stability
A29 - Status (of implementation)
A34 – Risk (of implementation)
A36 - Key Driving Need or Requirement (KDN/KDR)

3.4 C13 - COMPREHENSIBLE

Definition:

The set of need statements and resulting requirement statements must be written such that it is clear as to what is expected of the entity and its relation to the system of which it is a part.

Rationale:

An agreement is difficult to enact unless both parties are clear on the exact obligation and the expected outcome(s) as a result of the realization of the entity the set of needs and requirements represents. These sets must therefore be written such that the relevant audience can understand what is being communicated by the individual needs and requirements as well as the set of needs and requirements.

To be comprehensible, the set of needs or requirements must be complete, correct, consistent, feasible, verifiable and not include any unnecessary needs or requirements. Incorrect information for a set of needs or requirements could mean there are needs or requirements in the set that are unnecessary.

A set of requirements is verifiable if and only if all individual requirements in the set are verifiable.

Guidance:

Information needed to understand the context of the needs and requirements should be included with the set. This includes useful information attached to the set that defines the context defined during scope definition for the SOI as well as attributes of the needs and requirements such as rationale. This information is commonly included in the introductory material at the beginning of a documented set of needs or requirements.

Rules that help establish this characteristic:

R4 - /Accuracy/UserDefinedTerms
R18 - /Singularity/SingleSentence
R36 - /UniformLanguage/UseConsistentTerms
R37 - /UniformLanguage/DefineAcronyms
R38 - /UniformLanguage/AvoidAbbreviations
R39 - /UniformLanguage/StyleGuide
R40 - /Modularity/RelatedRequirements
R41 - /Modularity/Structured

Attributes that help establish this characteristic:

A1 - Rationale

3.5 C14 - ABLE TO BE VALIDATED

Definition:

It must be able to be proven that the set of needs will lead to the achievement of the product goals and objectives, stakeholder expectations, risks, and concepts within the constraints (such as cost, schedule, technical, legal and regulatory compliance) with acceptable risk.

It must be able to be proven that the set of requirements will lead to the achievement of the needs and higher level requirements within the constraints (such as cost, schedule, technical, and regulatory compliance) with accepable risk.

Rationale:

The transformation must be formal and able to be validated, not just for individual needs and requirements, but also for the sets of needs and requirements. It must be able to be shown at any life cycle stage of the system development that achievement of the set of needs will result in the concepts from which they were transformed and the achievement of the set of requirements will result in meeting the set of needs from which they were transformed.

Design validation is making sure the design and resulting specifications will result in a system that meets its intended purpose/use in its operational environment as defined by the set of needs. Thus design validation goes back to the needs to make sure they have been met—as a set.

System validation is making sure the built and verified system meets its intended purpose/use in its operational environment as defined by the set of needs. Thus system validation goes back to the needs to make sure they have been met—as a set. See Section 1.7 for more details on needs, requirements, design, and system validation.

For a set of needs or requirements to be validated, the set must be complete, correct, consistent, feasible, and comprehensible.

Guidance:

The system life-cycle operational scenarios, concepts, and use cases—from which the needs were transformed and the requirements were transformed—can be used as test cases to validate the requirement set, the design, and build or coded system. In other words, the requirement set can be validated to meet the needs by running tests based on the use cases or operational scenarios developed during scope definition activities. Doing so will result in proof that the intended use, goals, and objectives and stakeholder expectations have been met within the agreed-to drivers and constraints with acceptable risk.

For the set of needs, ask the questions: “Do the needs and set of needs clearly and correctly communicate the agreed-to concepts, constraints, and stakeholder expectations?” or “Have we correctly and completely capture the needs the system must address? For the set of requirements ask the questions: “Will the entity developed by this set of requirements satisfy the needs?” “Are we building the right thing?”

Rules that help establish this characteristic:

R3 - /Accuracy/SubjectVerb
R4 - /Accuracy/UseDefinedTerms
R36 - /UniformLanguage/UseConsistentTerms
R37 - /UniformLanguage/DefineAcronyms
R38 - /UniformLanguage/AvoidAbbreviations
R39 - /UniformLanguage/StyleGuide
R41 - /Modularity/Structured

4 RULES FOR NEED AND REQUIREMENT STATEMENTS AND SETS OF NEEDS AND REQUIREMENTS

This section defines the rules for individual need and requirement statements and sets of needs and requirements that help them to be formulated such that they will have the characteristics defined in the previous sections. Included with each rule is an explanation of the rule, examples of the application of the rule with a trace to the characteristics that are supported by the rule.

Note: Because need statements are written from the perspective of the stakeholders at a higher level of abstraction, some of the rules in this section may not always apply. The key is that the need statements have the characteristics defined earlier that are appropriate to the level of abstraction at which the need statements are written. For example, rules that address the verifiability or completeness of a requirement statement probably will not apply to a need statement.

In addition to the rules in this section, the reader is encouraged to follow the principles of good technical writing (as they apply to a ne3ed or requirement statement) such as those outlined in the Simplified Technical English (STE) specification (ASD-STE100).

4.1 ACCURACY

4.1.1 R1 - /ACCURACY/SENTENCESTRUCTURE

Use a structured sentence.

Elaboration:
The need or requirements statement must be in the form of a complete sentence, the simplest form of which is: <subject> <verb> <object>. The subject is essential because it is the entity undertaking the action; the verb is essential because it is the action being performed; and the object is essential because it must be clear as to which entity is being acted upon. The subject and objects are therefore normally entities and the verb is “shall” as described in Section 1. Use of “shall” makes it clear that what is being communicated is formal, the statement is legally binding and will be verified. Currently, no other form, other than textual “shall” requirement statements, have been shown to meet these characteristics. Some organizations may use other verbs such as “must”, “will”, “should”, or “may”—regardless, whichever verb is used, the mandatory, contractually binding nature of the action must be defined somewhere in the terms and conditions of the contract. ISO/IEC 29148 states that a more-complete, typical sentence form for a functional requirement is: When <condition clause>, the <subject clause> shall <action verb clause> <object clause> <optional qualifying clause>. See Appendix B for additional example patterns using clauses. There are a number of agreed types of condition: *Event-driven requirements:* When <optional preconditions/trigger> the <system name> shall <system response>.

In the event of <specific event> the <system name> shall <system response>.

Behavior driven requirements:

If <optional preconditions/trigger>, then the <system name> shall <system response>.

State-driven requirements:

While <entering, exiting or in a specific state (or mode)> the <system name> shall <system response>.

Note: While ISO/IEC 29148 shows the conditional phrase at the beginning of the sentence, some organizations prefer to put the conditional phrase at the end of the sentence. Organizations need to define the desired form in their standards, guides, and processes and then be consistent in the form being used. As long as the result is clear and unambiguous, either form is acceptable.

See also R18.

Examples:

Condition examples:

Event-driven requirements:

When *Continuous_Ignition is commanded by the Aircraft*, the *Control_System* shall *switch on Continuous_Ignition.*

In the event of a fire, the Security_System shall Unlock the Fire_Escape_Doors.

Behavior driven requirements*:*

If *the Computed_Airspeed Fault_Flag is set,* [then] the *Control_System* shall *use Modelled_Airspeed.*

State-driven requirements:

While *the Aircraft is In-flight*, the *Control_System* shall *Maintain Engine_Fuel_Flow at greater than XX lbs/sec.*

Qualification examples:

When *in the 'ON' state*, the *System* shall *display the Time*, *without obscuring the Work_Space*.

Characteristics that are established by this rule:

C3 - Unambiguous
C7 - Verifiable

4.1.2 R2 - /ACCURACY/USEACTIVEVOICE

Use the active voice in the main sentence structure of the need or requirement statement with the responsible entity clearly identified as the subject of the sentence.

Elaboration:

The active voice requires that the entity performing the action is the subject of the sentence. This is important in writing needs and requirements since the onus for satisfying the requirement is on the subject, not the object of the statement. If the entity responsible for the action is not identified explicitly, it is unclear who or what should perform the action making verification of that requirement very difficult. Including the entity in the subject also helps ensure the requirement refers to the appropriate level consistent with the entity name (see R3).

Often when the phrase "shall be" is used, the statement is in the passive voice.

Examples:
Unacceptable: The Identity of the Customer shall be confirmed. *{This is unacceptable because it does not identify the entity that is responsible/accountable for confirming the identity.}* *Acceptable*: The Accounting_System shall confirm the Customer_Identity. *{Note that" Accounting_System", "confirm", and "Customer_Identity" must be defined in the glossary since there are a number of possible interpretations of these terms.}* *Unacceptable*: The Audio shall be recorded by the System. *{This is unacceptable because the entity that is responsible/accountable for recording the audio is at the end of the sentence rather than the beginning.}* *Unacceptable*: The Audio shall be recorded. *{This is unacceptable because the entity that is responsible/accountable for recording the audio is not stated.}* *Acceptable*: The System shall record the Audio_Feed. *{Note that "Audio_Feed" must be defined in the glossary.}*

Characteristics that are established by this rule:
C2 - Appropriate C3 - Unambiguous C7 - Verifiable

4.1.3 R3 - /ACCURACY/SUBJECTVERB

Ensure the subject and verb of the need or requirement statement are appropriate to the level to which the need or requirement refers.

Elaboration:
<u>Subject</u> The subject of a need or requirement statement must be appropriate to the level to which it *refers*. Requirements referring to the business management level therefore have the form "The <business management> shall …"; those referring to the business operations level have the form "The <business operations> shall …"; those referring to the system level have the form "The <system> shall …"; requirements referring to the subsystem level have the form "The <subsystem> shall ..."; and requirements referring to the component level have the form "The <component> shall ...". As a general rule, need and requirement statements stated at a particular level should refer only to that level—that is, the business management requirements refer to the business management level, business operations requirements refer to the business operations level, system requirements refer to the system level, subsystem requirements refer to the subsystem level, and component requirements refer to the component level. In some cases, however, a higher level may wish to be prescriptive at a lower level. For example, it may be important for a business to mandate that the new aircraft under development must use a particular engine (perhaps for support reasons) in which case they may make a statement at the business level that refers to an entity at the subsystem level. Therefore, any level of the organization can state (at that level) requirements that refer to two levels: to that level, as well any lower level the requirement applies (should there be a good reason to do so). When this is the case, the prescriptive allocated requirement is treated as a constraint – a higher level requirement. In these cases, the lower level will then write a level specific child requirement and trace that child requirement to its parent or source.

Consequently, regardless of the level at which the requirement is *stated*, the subject of a requirement statement must be appropriate to the level to which it *refers*. To continue our aircraft example above, although the majority of requirements at the business management level will begin with "The ACME Aircraft Company shall …", the business may therefore wish to state at the business management level a requirement stating that all aircraft developed by the organization shall use an engine with specific characteristics. For a specific aircraft, children requirements will be written at the appropriate level that implement the intent of the business management generic requirement. For the level dealing with the engine specifically, the subsystem level child requirement would begin "The Engine shall …" and trace back to the business management level constraint as the parent or source.

Verb

Similarly, the verb of a need or requirement statement must be appropriate to the subject of the need or requirement at the level it is stated. For needs the verbs such as "support", "process", "handle", "track", "manage", and "flag" may be appropriate. However, they are too vague for requirement statements which may therefore be ambiguous and unverifiable. At the business management level, for example, the use of a verb such as "safe", may be acceptable as long as it is unambiguous at that level, decomposed at the lower levels, and is verifiable at those levels.

Examples:

Subject examples*:*

*Business management requirements have the form "*The <business> shall …"—for example, "ACME_Transport shall …"*.*

Business operations requirements on personnel roles have the form: "The <personnel role> shall …"*—for example,* "The Production_Manager shall …"*;* "The Marketing_Manager shall …"*.*

System level needs have the form "The stakeholders need the system to ……"

System requirements have the form "The <system> shall ..."*—for example,* "The Aircraft shall …"

Subsystem level needs have the form "The stakeholders need the subsystem to ……"

Subsystem requirements have the form "The <subsystem> shall ..." *–for example, once the subsystems are defined:* "The Engine shall …"; "The Landing_Gear shall …".

Verb examples*:*

System level stakeholder need: "The stakeholders need the system to process data received from [other system] XYZ."

System level requirement: "The system shall [process] data having the characteristics defined in *[Other system] Interface Definition XYZ*."

Through analysis, the verb/function "process" could be decomposed into sub functions such as "receive", "store", "calculate", "report", and "display". Then a decision needs to be made regarding the level at which these sub functions are to be stated. If more than one subsystem is involved in any one of the sub functions, that requirement should be communicated at the system level and allocated to the applicable subsystems. If a sub function is to be implemented by a single subsystem, then the sub function requirement should be communicated at the subsystem level and traced back to the parent requirement from which it was decomposed.

Unacceptable system requirement: "The User shall ……." *{This is unacceptable because the requirement should be on the system, not the user or operator of the system. This wording is often the result of writing requirements directly from user stories or resulting need statements, without doing the transformation of the use case or user need into a system requirement. Ask, what does the system have to do so that the user need can be achieved?}*

Acceptable: "The <system> shall ……."

Characteristics that are established by this rule:
C2 - Appropriate C3 - Unambiguous C7 - Verifiable C10 - Complete C14 - Able to be Validated

4.1.4 R4 - /ACCURACY/USEDEFINEDTERMS

Define terms.

Elaboration:
Most languages are rich with words having several synonyms, each with a subtly different meaning based on context. In needs and requirements statements, shades of meaning will most likely lead to ambiguity and to difficulty in verification and validation. Define the term in some form of ontology, including a glossary, data dictionary, or similar artifact that allows the reader of a need or requirement to know exactly what the writer meant when the word was chosen. The meaning of a term should be the same every time the word is used no matter the work product, SE tool, or artifact being developed across all life cycle stages. A standard should be agreed upon to make the use of glossary terms identifiable in the need and requirements text statements; for example, glossary items may be capitalized and multiple words in single terms joined by an underscore (e.g. “Current_Time”). This is essential for consistency to avoid using the word with its general meaning. This is the convention used in the examples in this section. This standard should be implemented and enforced within all SE tools used by the project to help ensure consistency. Definitions of terms need to be agreed to, documented, and used consistently throughout the project and all artifacts developed during all life cycle activities.

Examples:
Unacceptable: The system shall display the current time. *{This is unacceptable because it is ambiguous--what is “current”, in which time zone, to what degree of accuracy, in what format?}* *Acceptable*: The system shall display the Current_Time. *{Note that “Current_Time” must then be defined in the glossary in terms of accuracy, format, and time zone.}*

Characteristics that are established by this rule:
C3 - Unambiguous C7 - Verifiable C11 - Consistent C13 - Comprehensible C14 - Able to be Validated

4.1.5 R5 - /ACCURACY/USEDEFINITEARTICLES

Use definite article “the” rather than the indefinite article “a.”

Elaboration:
The definite article is “*the*”; the indefinite article is “*a*.” When referring to entities, use of the indefinite article can lead to ambiguity. For example, if the need or requirement refers to “a user” it is unclear whether it means any user or one of the defined users for which the system has been

designed. This causes further confusion in verification and validation—for example, babies are arguably users of baby food, but the system would fail if the test agency sought to verify or validate that a baby could order, receive, open, and serve (or even independently consume) baby food. On the other hand, if the requirement refers to "the User", the reference is explicitly to the nature of the user defined in the glossary—in the baby food example, the "User" is presumably the adult responsible for feeding the baby.

Examples:

Unacceptable: The system shall provide a time display. *{This is unacceptable for a requirement because it is ambiguous--will any time display do? Is a one-off display of the time satisfactory? The writer's intention was most likely that they wanted the system to display continuously the current time, yet if the developer provided a constant display of "10:00 am" (or even a one-off display of any time), they could argue (albeit unreasonably) that they have met the requirement; yet they would have clearly failed to meet the customer's need and intent.}{As a stakeholder need statement: "The stakeholders need the system to provide a time display." The statement is acceptable. As part of the transformation process the requirement writers will remove the ambiguity resulting in the following requirement statement.]*

Acceptable: The system shall display the Current_Time. *{Note that "Current_Time" must be defined in the glossary since there are a number of possible meanings and formats of the more-general term "current time." There may also be other requirements addressing where time needs to be displayed, accuracy, etc. if not addressed in the glossary or data dictionary.}*

Exceptions and relationships:

The purpose of this rule is to avoid the ambiguity that arises because "a" or "an" is tantamount to saying "any one of". In some cases, however, the use of an indefinite article is not misleading. For instance, "… with an accuracy of less than 1 second" allows the phrase to read more naturally and there is no ambiguity because of the accuracy quoted.

Characteristics that are established by this rule:

C3 - Unambiguous
C7 - Verifiable

4.1.6 R6 - /ACCURACY/UNITS

Use appropriate units when stating quantities.

Elaboration:

All numbers should have units of measure explicitly stated in terms of the measurement system used or the thing the number refers.

Within a project, a common measurement system must be used consistently. For example, don't mix both US and metric units of measure within any of the project's artifacts.

There are three primary measurement systems: British imperial, US, Metric

For temperatures the following are used: celsius, fahrenheit, or kelvin, etc.

Examples:

Unacceptable: The Circuit_Board shall have a storage temperature less than 30 degrees. *{This is unacceptable because the units used are incomplete.}*

Acceptable: The Circuit_Board shall have a storage temperature of less than 30 degrees Celsius.

Unacceptable: The system shall establish connection to at least 4 in less than or equal to 10 seconds. *{This is unacceptable because the units used are incomplete. "4" of what?}*

Acceptable: The system shall establish communications to greater than 4 satellites in less than or equal to 10 seconds. *{Note that the term "communications" is acceptable at the business level, but would need further elaboration at lower levels so that the requirement is decomposed and verifiable. E.g. frequency, type of communications (voice? data?), quality, bandwidth, etc.}*

Characteristics that are established by this rule:
C3 - Unambiguous C4 - Complete C7 - Verifiable C8 - Correct

4.1.7 R7 - /ACCURACY/AVOIDVAGUETERMS

Avoid the use of vague terms.

Elaboration:
Avoid words that provide vague quantification, such as "some", "any", "allowable", "several", "many", "a lot of", "a few", "almost always", "very nearly", "nearly", "about", "close to", "almost", and "approximate", . Avoid vague adjectives such as "ancillary", "relevant", "routine", "common", "generic", "significant", "flexible", "expandable", "typical", "sufficient", "adequate", "appropriate", "efficient", "effective", "proficient", "reasonable" and "customary." Vague adjectives can lead to ambiguous, unverifiable requirements that do not reflect accurately the stakeholder expectations. Adverbs qualify actions in some way and are particularly troublesome. Avoid vague adverbs, such as "usually", "approximately", "sufficiently", and "typically". Vague adverbs can lead to ambiguous, unverifiable requirements that do not reflect accurately the stakeholder expectations. As a general rule, words that end in "-ly" often result in ambiguity.

Examples:
Unacceptable: The Flight_Information_System shall usually be on line. *{This is unacceptable because "usually" is ambiguous - is availability what is meant?}* *Acceptable*: The Flight_Information_System shall have an Availability of greater than xx% over a period of greater than yy hours. *{Note that "Availability" must be defined in the glossary since there are a number of possible ways of calculating that measure.}* *Unacceptable*: The Flight_Information_System shall display the Tracking_Information for relevant aircraft. *{This is unacceptable because it does not make explicit which aircraft are relevant. Additionally, the statement allows the developer to decide what is relevant; such decisions are in the province of the customer, who should make the requirement explicit.}* *Acceptable*: The Flight_Information_System shall display the Tracking_Information of each Aircraft located less than or equal to 20 kilometers from the Airfield. *{Now it is clear for which aircraft the information needs to be displayed. Note that "Aircraft", "Tracking_Information", and "Airfield" must be defined in the glossary.}*

Exceptions and relationships:
R3 points out that the use of a verb such as “safe”, may be acceptable at the business management or operations level as long as it is unambiguous at that level, decomposed at the lower levels, and is verifiable at the level stated. Similarly, some vague adjectives may be allowable at the business management or operations level, providing they are not ambiguous at that level. For example, it may be appropriate for the business to state “The system shall provide a user-friendly interface.” Or “The stakeholders need the system to provide a user-friendly interface.” Although ‘”user-friendly” will need to be decomposed and verified at lower levels before it is useful, it may be arguably sufficiently clear at the business management or operations level or in a higher level need statement.

Characteristics that are established by this rule:
C3 - Unambiguous C4 - Complete C7 - Verifiable

4.1.8 R8 - /ACCURACY/NOESCAPECLAUSES

Avoid escape clauses.

Elaboration:
Escape clauses give an excuse to the developer of the system at lower levels not to bother with a need or requirement. From a contracting standpoint, needs or requirements with these phrases could therefore be interpreted as being optional even if communicated in a “shall” requirement statement. Such clauses provide vague conditions or possibilities, using phrases such as “so far as is possible”, “as little as possible”, “where possible”, “as much as possible”, “if it should prove necessary”, “if necessary”, “to the extent necessary”, “as appropriate”, “as required”, “to the extent practical”, and “if practicable.” Escape clauses can lead to ambiguous needs that can’t be validated and are open to interpretation and that do not reflect accurately concepts from which they were transformed. Escape clauses can lead to ambiguous, unverifiable requirements that are open to interpretation and that do not reflect accurately the needs from which they were transformed.

Examples:
Unacceptable: The GPS shall, *where there is sufficient space*, display the User_Location. *{This is unacceptable because whether there is sufficient space is vague, ambiguous, and unverifiable. The requirement is clearer without the escape clause.}* *Acceptable*: The GPS shall display the User_Location. *{Note that “GPS” and ”User_Location” must be defined in the glossary. Specific performance requirements need to be defined as well such as within what time, format, and accuracy.}* *Unacceptable*: The Television shall, *to the extent necessary*, conform to the requirements in Section a.1 of Standard XYZ. *{This is unacceptable because it is unclear who is to determine the meaning of ‘necessary’. The requirement is clearer without the escape clause.}* *Acceptable*: The Television shall conform to the requirements in Section a.1 of Standard XYZ. *{Note that this form of requirement means that each requirement of Standard XYZ is to be met—if only a subset of a standard or regulation is required, that subset must be either listed explicitly or implementing requirements derived that meet the intent of the specific requirements in the standard and regulation, with a trace to the source or parent requirement.}*

Characteristics that are established by this rule:
C3 - Unambiguous C4 - Complete C7 - Verifiable

4.1.9 R9 - /ACCURACY/NOOPENENDED

Avoid open-ended clauses.

Elaboration:
Open-ended clauses imply there is more required without stating exactly what. Avoid non-specific phrases such as “including but not limited to”, “etc.” and “and so on.” Open-ended clauses can lead to ambiguous, unverifiable needs and requirements that do not reflect accurately the stakeholder’s expectations and needs and can create ambiguity in the mind of the reader. Use of open-ended clauses also violates the one-thought rule that leads to the singular characteristic. If more cases are required, then include additional needs and requirements that explicitly state those cases. Depending on the contract type (fixed price versus level of effort or cost plus) open-ended needs and requirement statements can lead to serious interpretation problems concerning what is in or out of scope of the contract.

Examples:
Unacceptable: The ATM shall display the Customer Account_Number, Account_Balance, and so on. *{This is unacceptable because it contains an opened list of what is to be displayed.}* *Acceptable*: *{Split into as many requirements as necessary to be complete. Note that the types of customer information to be displayed needs to be defined in the glossary.}:* The ATM shall display the Customer Account_Number. The ATM shall display the Customer Account_Balance. The ATM shall display the Customer Account_Type. The ATM shall display the Customer Account_Overdraft_Limit. *Note: Some may feel that a bulleted list or table is acceptable. While, from a readability perspective, this may be true, from a requirement management perspective, each item in the bulleted list is still a requirement. Because of this, is better to write individual requirement statements as was done in the example above. This allows allocation, traceability, and verification activities to be specific to the single item. See also R18 and R22.*

Characteristics that are established by this rule:
C3 - Unambiguous C4 - Complete C5 - Singular C7 - Verifiable

4.2 CONCISION

4.2.1 R10 - /CONCISION/SUPERFLUOUSINFINITIVES

Avoid superfluous infinitives.

Elaboration:
We sometimes see a need or requirement that contains more verbs than necessary to describe a basic action, such as “The system shall be designed to be able to ...” or “The system shall be designed to be capable of ...” rather than simply “The system shall ...” Think ahead to verification. If the system is able to, or is capable of, doing something one time but fails 99 times, the system has not met the requirement. Note that at the enterprise and business levels, requirements for an entity to “provide a capability” are acceptable. Where capability is made up of people, processes, and products; these requirements will be decomposed to address the people aspects (skill set, training, roles, etc.), processes (procedures, work instructions, etc.); and products (hardware and software systems). The parent requirement will be allocated to the people, processes, and products (SOI), as appropriate. When the resulting requirement sets are implemented for all three areas, the capability will exist to meet the need. This is also true for needs at the system or system element levels. The stakeholders may have a need for the system to “be able to” or “have the capability to” do something. For example: “The stakeholders need the system to provide the capability for users to [do something] ...”. As part of the transformation process the requirement writer will determine what the system needs to do to provide that capability. The resulting well-formed requirements will then define what the system needs to do to supply that capability in unambiguous and verifiable language. If the intent of using “be able to” or “be capable of” type wording is to communicate the system will only need to do the required action based on some condition, trigger, or state, then it is best to state the condition, trigger, or state as part of the requirements. When this is the case, refer to R1.

Examples:
Unacceptable: The Weapon_Subsystem shall *be able to* store the location of each Ordnance. *{This is unacceptable because it contains the superfluous infinitive “be able to”. However, this is acceptable for the stakeholder need: “The stakeholders need the Weapon_Subsystem to be able to store the location of each Ordnance.”}* *Acceptable*: The Weapon_Subsystem shall store the Location of each Ordnance. *{Note that the terms “Weapon_Subsystem”, “Location”, and “Ordnance” must be defined in the glossary.}*

Characteristics that are established by this rule:
C3 - Unambiguous C7 - Verifiable

4.2.2 R11 - /CONCISION/SEPARATECLAUSES

Use a separate clause for each condition or qualification.

Elaboration:
Each need or requirement should have a main verb describing a basic function or need. If appropriate, the main sentence may then be supplemented by clauses that provide conditions or

qualifications (performance values or constraints). A single, clearly identifiable clause should be used for each condition or qualification expressed.

When using clauses, make sure the clause does not separate the object of the sentence from the verb. See also R1 and Appendix B, Patterns.

Examples:

Unacceptable: The Navigation_Beacon shall provide Augmentation_Data at an accuracy of less than or equal to 20 meters to each Maritime_User during Harbor_Harbor_Approach_Maneuvering (HHAM). {*This is unacceptable because it inserts a phrase in such a way that the object of the sentence is separated from the verb*.}

Acceptable: The Navigation_Beacon shall provide Augmentation_Data to each Maritime_User engaged in Harbor_Harbor_Approach_Maneuvering (HHAM), at an accuracy of less than or equal to 20 meters. {*This places the basic function in an unbroken clause followed by the sub-clause describing performance*.}

{Note that: "Navigation_Beacon", "Maritime_User", and "Harbor_Harbor_Approach_Maneuvering (HHAM)" must be defined in the glossary.}

Characteristics that are established by this rule:

C3 - Unambiguous

4.3 NON-AMBIGUITY

4.3.1 R12 - /NONAMBIGUITY/CORRECTGRAMMAR

Use correct grammar.

Elaboration:

We interpret language based on the rules of grammar. Incorrect grammar leads to ambiguity and clouds understanding. This is especially true when the recipient of the need or requirement statement is working in a second language relying on specific rules of grammar. If these rules are not followed, that person may misinterpret the meaning of the need or requirement statement.

Care must be taken when translating needs and requirements from one language to another and when sentence structure differs depending on the language in which the original need or requirement statement was written. Punctuation varies from language to language and even between dialects of a given language. Be very cautious when need and requirement statements must be translated. An interesting exercise is to translate a requirement from one language to another and translate the result back to the original language.

Examples:

Unacceptable: The Weapon_Subsystem shall *storing* the location of all ordnance. *{This is unacceptable because the grammatical error leads to uncertainty about the meaning.}*

Acceptable: The Weapon_Subsystem shall store the location of all Ordnance. *{Note that "Ordnance" must be defined in the glossary to be explicit about the types of weapons and ammunition.}*

Unacceptable: When in the ACTIVE State, the Record_Subsystem shall display each of the Names of the Line_Items, without obscuring the User_ID. *{This is unacceptable because the*

grammatical error involving the inappropriate placement of "each of"—it is most likely that a Line_Item has only one name.}

Acceptable: When in the ACTIVE State, the Record_Subsystem shall display the Name of each Line_Item, without obscuring the User_ID.

Characteristics that are established by this rule:
C3 - Unambiguous
C9 - Conforming

4.3.2 R13 - /NonAmbiguity/CorrectSpelling

Use correct spelling.

Elaboration:
Incorrect spelling can lead to ambiguity and confusion. Some words may sound the same but, depending on the spelling, will have entirely different meaning. For example, "red" versus "read" or "ordinance" versus "ordnance." In these cases, a spell checker will not find the error.

In addition to misspelling, this rule also refers to the proper use of:

- Capital letters in acronyms: avoid "SYRD" and "SyRD" in the same specification.
- Capital letters in other non-acronyms concepts: avoid "Requirements Working Group" and "Requirements working group" in the same specification.
- Proper use of hyphenation: "non-functional" versus "nonfunctional." Often hyphenation is used when two related words are used as adjectives, but is not used when used as a noun.

Examples:
Unacceptable: The Weapon_Subsystem shall store the location of all *ordinance*. *{This is unacceptable because the word "ordinance" means regulation or law. It is unlikely that the Weapon_Subsystem is interested in the location of ordinance (regulations). In the context of a weapon system, what the authors meant to use is "ordnance" as in weapons and ammunition, not "ordinance".}*

Acceptable: The Weapon_Subsystem shall store the Location of all Ordnance. *{Note that "Location" and "Ordnance" must be defined in the glossary to be explicit about the types of weapons and ammunition.}*

Characteristics that are established by this rule:
C3 - Unambiguous

4.3.3 R14 - /NonAmbiguity/CorrectPunctuation

Use correct punctuation.

Elaboration:
Incorrect punctuation can cause confusion between sub-clauses in a need or requirement statement. Note also that the more punctuation in the need or requirement statement, the greater the opportunity for ambiguity.

Examples:

Unacceptable: The Navigation_Beacon shall provide Augmentation_Data to each Maritime_User, engaged in Harbor_Harbor_Approach_maneuvering (HHA) at an accuracy of 20 meters or less at least 99.7% of the time. *{This is unacceptable because the incorrectly placed comma in this sentence confuses the meaning, leading the reader to believe that the accuracy is related to the maneuver rather than to the augmentation data.}*

Acceptable: The Navigation_Beacon shall provide Augmentation_Data to each Maritime_User engaged in Harbor_Harbor_Approach_maneuvering (HHA), at an accuracy of less than or equal to 20 meters greater than 99.7% of the time.

{The positioning of the comma now makes it clear that the accuracy and availability relate to the data.}

Characteristics that are established by this rule:

C3 - Unambiguous

4.3.4 R15 - /NONAMBIGUITY/LOGICALCONDITION

Use a defined convention to express logical expressions.

Elaboration:

Define a convention for logical expressions such as “[X AND Y]”, “[X OR Y]”, [X XOR Y]”, “NOT[X OR Y]”.

As with the other rules and characteristics, we want to keep requirement statements as one thought with singular statements. Thus, we avoid using “and” when it involves tying two thoughts together. However, it is acceptable to use “AND”, “OR”, “XOR”, and “NOT” in a logical sense when talking about conditions to which the verb applies. All logical expressions decompose to either “true” or “false”, resulting in a singular statement.

Examples of conventions:

1. Place conjunctions in italics or in all capitals (AND, OR, XOR, NOT) to indicate that the author intends the conjunction to play a role in a condition.
2. Place conditions within square brackets, also using the brackets to control their scope.

For example, “[X AND Y].”

Further, use of “and/or” is non-specific and therefore ambiguous. The most common interpretation of the expression “and/or” is as an inclusive or: either X or Y or both. If that is the intention, it should be written as two requirements, each of which can be verified separately.

Note that caution should be used when including logical expressions in design input requirements. In many cases the use of logical expressions is more appropriate to design output specifications/requirements. Input requirements should focus on why the logical actions are needed – prevent something bad from happening, for example,

Also see R19 and R20.

Examples:

Unacceptable: The Engine_Management_System shall disengage the Speed_Control_Subsystem when the Cruise_Control is engaged, and the Driver applies the Accelerator. *{This is unacceptable because of the ambiguity of “and” could be confused with combining two separate thoughts. Instead use the form of a logical expression [X AND Y].}*

Acceptable: The Engine_Management_System shall disengage the Speed_Control_Subsystem, when [the Cruise_Control is engaged AND the Driver applies the Accelerator].

Exceptions and relationships:

While R21: /Singularity/AvoidParentheses states that parentheses or brackets are to be avoided within general sentence structure, this rule suggests that brackets may be used as part of a convention to avoid ambiguity in logical conditions.

Characteristics that are established by this rule:

C3 - Unambiguous

4.3.5 R16 - /NonAmbiguity/AvoidNot

Avoid the use of "not."

Elaboration:

The presence of "not" in a need or requirement statement implies "not ever", which is impossible to verify in a finite time.

Rewriting the need or requirement statement to avoid the use of "not" results in a need or requirement statement that is clearer and is verifiable or able to be validated.

Examples:

Unacceptable: The <system> shall not fail. *{This is unacceptable because verification of the requirement would require infinite time.}*

Acceptable: The <system> shall have an Availability of greater than or equal to 95%. Or The <system> shall have a Mean Time Between Failures (MTBF) of 6 months.

Unacceptable: The <system> shall not contain mercury. *{This is unacceptable because verification of the requirement would require infinite precision. In addition, the real requirement may not be stated, for example, the real concern may be the use of toxic materials, not just mercury. If that is the case, it may be best to reference a standard from a governmental agency concerning allowable exposures to a list of common toxic materials.}*

Acceptable: The <system> shall limit metallic mercury exposures to less than or equal 0.025 mg/m_3 over an 8-hour workday.

Exceptions and relationships:

It may be reasonable to include "not" in a requirement when the logical "NOT" is implied—for example when using not [X or Y]. In that case, however, in accordance with Rule 16, it may be better to capitalize the "NOT" to make the logical condition explicit: NOT [X or Y]. There may be other cases such as "The <system> shall not be red in color." Which is stating a constraint and is verifiable, as long as the range of shades of red is stated (RBG rr,bb,gg range or a "name" of red in some standard).

Characteristics that are established by this rule:

C3 - Unambiguous
C7 - Verifiable

4.3.6 R17 - /NONAMBIGUITY/OBLIQUE

Avoid the use of the oblique ("/") symbol.

Elaboration:
The oblique symbol (“/”), or “slash”, has so many possible meanings that it should be avoided. The slash symbol (like the construct “and/or” discussed in R15) can lead to ambiguous need and requirement statements that do not reflect accurately the true customer needs or concept from which the needs were derived. An exception to this rule is where the oblique symbol is used in SI units (for example “km/h”). Also see R19.

Examples:
Unacceptable: The User_Management_System shall Open/Close the User_Account in less than 1 second. *{This is unacceptable because it is unclear as to what is meant: open, close, or both?}* *Acceptable (Split into two requirements)*: The User_Management_System shall Open the User_Account in less than 1 second. The User_Management_System shall Close the User_Account in less than 1 second. *Unacceptable*: The Engine_Management_System shall disengage the Speed_Control_Subsystem when the Clutch is disengaged and/or the Driver applies the Brake. {*This is unacceptable because of the use of “and/or.” If “and” is meant, split the two thoughts into separate requirements. If “or” is meant, write the requirement as an exclusive “or.”}* *Acceptable as two requirements*: - The Engine_Management_System shall disengage the Speed_Control_Subsystem when the Clutch is disengaged. - The Engine_Management_System shall disengage the Speed_Control_Subsystem when the Driver is applying the Brake. *Acceptable as one requirement if exclusive or is intended*: - The Engine_Management_System shall disengage the Speed_Control_Subsystem when [the Clutch is disengaged OR the Driver is applying the Brake].

Characteristics that are established by this rule:
C3 - Unambiguous C7 - Verifiable

4.4 SINGULARITY

4.4.1 R18 - /SINGULARITY/SINGLESENTENCE

Write a single sentence that contains a single thought conditioned and qualified by relevant sub-clauses.

Elaboration:
Based on the concepts of allocation, traceability, validation, and verification, requirement statements must contain a single thought allowing that single thought to be allocated, the resulting single thought children requirements to trace to their allocated parent, requirements to trace to a single thought source, and allowing validation and verification of the single thought.

Sometimes a need or requirement statement is only applicable under a specific condition or multiple conditions. See R1, R28, and A6.

If multiple actions are needed for a single condition, each action should be repeated in the text of a separate need or requirement statement along with the triggering condition, rather than stating the condition and then listing the multiple actions to be taken.

Also avoid stating the condition or trigger for an action in a separate sentence. Instead write a simple affirmative declarative sentence with a single subject, a single main action verb and a single object, framed and qualified by one or more sub-clauses. See R1. Compound sentences are to be avoided.

Often when there are multiple sentences for one requirement, the writer is using the second sentence to communicate the conditions for use or rationale for the requirement for the first sentence. This practice is not acceptable – rather include this information in the attribute A1 - *Rationale* as part of the requirement expression as discussed in Section 5.1.1.

Examples:

Unacceptable: When in the ACTIVE State, the Record_Subsystem shall display the Name of each Line_Item *and* shall record the Location of each Line_Item, without obscuring the User_ID. *{This is unacceptable because the sentence contains two requirements and the qualification only applies to the first requirement, not the second.}*

Acceptable (Split into two requirements):

When in the ACTIVE State, the Record_Subsystem shall display the Name of each Line_Item, without obscuring the User_ID.

When in the ACTIVE State, the Record_Subsystem shall record the Location of each Line_Item.

{Note use of the glossary to define terms.}

Unacceptable: The Control_Subsystem will close the Inlet_Valve until the temperature has reduced to 85 °C, when it will reopen it. *{This is unacceptable because the sentence contains two event driven requirements. Additionally, the sentence contains two occurrences of the pronoun "it" ambiguously referring to different things (see R26), the term 'has reduced' is ambiguous, and the action verb must be 'shall', not 'will'.}*

Acceptable (Split into two requirements):

If the temperature of water in the Boiler is greater than 85 °C., the Control_Subsystem shall close the Inlet_Valve in less than 3 seconds

When the temperature of water in the Boiler is reduced to less than or equal to 85 °C., the Control_Subsystem shall open the Inlet_Valve in less than 3 seconds.

{Note use of the glossary to define terms.}

Unacceptable: In the event of a fire:

- Power to electromagnetic magnetic fire door catches shall be turned off.
- Security entrances shall be set to Free_Access_Mode.
- Fire escape doors shall be unlocked.

{This is unacceptable because the condition "in the event of a fire" is stated following a list of actions to be taken. Also, note the actions are written in passive voice which violates R2. Each action needs to be communicated in a separate requirement statement. – for multiple conditions for a single action see R28.}

Acceptable:
- In the event of a fire, the Security_System shall turn off power to electromagnetic magnetic fire door catches.
- In the event of a fire, the Security_System shall set security entrances to the Free_Access_Mode.
- In the event of a fire, the Security_System shall unlock fire escape doors.

Exceptions and relationships:

Every requirement should have a main clause with a main verb (R1). However, additional sub-clauses with auxiliary verbs or adverbs may be used to qualify the requirement with performance attributes.

Such sub-clauses cannot be verified in isolation, since they are incomprehensible without the main clause. Sub-clauses that need to be verified separately from others should be expressed as separate requirements.

For example, “The Ambulance_Control_System shall communicate Incident_Details to the Driver” is a complete, comprehensible statement with a single main verb. An auxiliary clause may be added to provide a constraint “The Ambulance_Control_System shall communicate Incident_Details to the Driver *while simultaneously maintaining communication with the Caller.”*

Similarly, if the requirement is to extinguish and dispose of a match as a single combined action, the requirement must ensure that both are verified the same time and allocated the same.

Note that, if performance attributes need to be verified separately, they should be expressed as sub-clauses in separate requirements.

Characteristics that are established by this rule:

C3 - Unambiguous
C4 - Complete
C5 - Singular
C7 - Verifiable
C9 - Conforming
C13 - Comprehensible

4.4.2 R19 - /SINGULARITY/AVOIDCOMBINATORS

Avoid combinators.

Elaboration:

Combinators are words that join clauses, such as “and”, “or”, “then”, “unless”, “but”, “as well as”, “but also”, “however”, “whether”, “meanwhile”, “whereas”, “on the other hand”, and “otherwise.” Their presence in a requirement usually indicates that multiple requirements should be written.

Exception: AND, OR, NOT can be used in need and requirement statements as logical conditions and qualifiers as stated in R15.

See also R16 and R17.

Examples:

Unacceptable: The user shall either be trusted or not trusted. *{This is unacceptable for a number of reasons. The intention is that a user should be classified in one of two ways but it is also a passive requirement written on the user rather than on the system (R2) and it is ambiguous: the requirement would still be met if the system took the option of treating all users as trusted.}*

Acceptable: The Security_System shall categorize each User as [EITHER Trusted OR Not_Trusted).

Acceptable: The Security_System shall categorize each User as one of the following: Trusted, Not_Trusted.

(At the subsystem or component specification level.)

Unacceptable: The "command the sidelights" function shall ensure the regulatory consistency of lights illumination *by ensuring* the illumination of sidelights is maintained during the illumination of the side lamps, or the head lights, or the fog lights or the rear fog lights or a combination of these lights. *{Again, this is unacceptable because it is non-singular, uses dubious grammar, contains elements of purpose, and is generally confusing.}*

Acceptable: The Control_Sidelights function shall illuminate the Sidelights while any combination of the following conditions are true: the Side_Lamps are illuminated, the Head_Lamps are illuminated, the Front_Fog_Lamps are illuminated, the Rear_Fog_Lamps are illuminated. *{Rules 16 and 17 have also been applied to remove combinators and clarify the exact conditions. Note that the use of "any" in this example is not ambiguous and thus is an exception to R24 and R32.}*

Characteristics that are established by this rule:

C3 - Unambiguous
C5 - Singular

4.4.3 R20 - /SINGULARITY/AVOIDPURPOSE

Avoid phrases that indicate the purpose of the need or requirement statement.

Elaboration:

The text of a need or requirement statement does not have to carry around extra baggage such as the purpose for its existence. Expressions of purpose are often indicated by the presence of phrases such as "……to", "in order to", "so that", and "thus allowing."

This extra information should be included in the rationale attribute (A1).

Examples:

Unacceptable: The Record_Subsystem shall display the Name of each Line_Item so that the Operator can confirm that it is the correct Item. *{This is unacceptable because the text "so that the Operator can confirm that it is the correct Item" is rationale.}*

Acceptable: The Record_Subsystem shall display the Name of each Line_Item. [The text "*so that the Operator can confirm that it is the correct Item" should be included in the rationale attribute.]*

Unacceptable: The Operation_Logger shall record each Warning_Message produced by the system. *{This is unacceptable because of the superfluous words "produced by the system".}*

Acceptable: The Operation_Logger shall record each Warning_Message.

Characteristics that are established by this rule:

C5 - Singular

4.4.4 R21 - /SINGULARITY/AVOIDPARENTHESES

Avoid parentheses and brackets containing subordinate text.

Elaboration:
If the text of a need or requirement statement contains parentheses or brackets, it usually indicates the presence of superfluous information that can simply be removed or communicated in the rationale. Other times, brackets introduce ambiguity. If the information in the parentheses or brackets aids in the understanding of the intent of the requirement, then that information should be included in the rationale attribute (A1). Conventions for the use of parentheses or brackets may be agreed upon for specific purposes, but such conventions should be documented at the start of the needs or requirements document.

Examples:
Unacceptable: The Control_Unit shall disconnect power from the Boiler when the water temperature exceeds 85 °C (usually towards the end of the boiling cycle.) *{This is unacceptable because of the parenthetical phrase as well as the ambiguous word "usually".}* *Acceptable*: The Control_Unit shall disconnect power from the Boiler when the water temperature is greater than 85 °C. *Note: for the above example, if this behavior is the result of a design decision, then the requirement needs to be communicated as a design output. The design input requirement should focus on why this behavior is needed – for example, prevent the boiler from over pressurizing and exploding.*

Exceptions and relationships:
While this rule states brackets are to be avoided, R15 indicates that brackets may be used as part of a convention for eliminating ambiguous conditions such as logical expressions.

Characteristics that are established by this rule:
C5 - Singular

4.4.5 R22 - /SINGULARITY/ENUMERATION

Enumerate sets explicitly instead of using a group noun to name the set.

Elaboration:
If a number of functions are implied, a need or requirement statement should be written for each. The use of a group noun to combine functions or entities is often ambiguous because it leaves membership of that group in doubt. Other issues include allocation, traceability, verification, and validation. As with the singularity characteristic C5, each member of the set could be allocated differently, have different children each of which needs to be individually verified and validated. It is almost always best to list all the members of the set as separate needs or requirements.

Examples:
Unacceptable: The thermal control system shall manage temperature-related functions. *{This is unacceptable because there is ambiguity regarding which functions are to be managed. The specific functions should be enumerated explicitly in separate requirement statements.}* *Acceptable (three separate requirements):* The Thermal_Control_System shall update the display of Current_System_Temperature every 10 +/- 1 seconds. The Thermal_Control_System shall maintain the Current_System_Temperature between 95°C and 98°C. The Thermal_Control_System shall store the history of Current_System_Temperature. *{Note use of the glossary to define terms.}*

Exceptions and relationships:
It is almost always better to enumerate sets but the rule may be softened at the higher levels of abstraction if the resulting requirement is sufficiently unambiguous. For example, at the business management level the business may state “ACME Consulting shall manage HR functions centrally.” or a system level need statement may state: “The stakeholders need the system to manage HR functions centrally.” Although this raises the question of which HR functions are being referred to, it is not useful to include a very long list at these levels of abstractions and it is often not necessary since the statement is sufficiently clear at the level at which it is stated. The detailed enumeration of functions will be undertaken at the business operations level, and the business management stakeholders should be comfortable that no function will be omitted as a result of not listing it at the business management level.

Characteristics that are established by this rule:
C2 – Appropriate C3 – Unambiguous C5 – Singular

4.4.6 R23 - /SINGULARITY/CONTEXT

When a need or requirement is related to complex behavior, refer to the supporting diagram or model.

Elaboration:
Sometimes it can be very difficult to express a complicated need or requirement in words, and it is better simply to refer to a diagram or model. An example is a requirement on voltage at turn on which involves a magnitude, rise time, over shoot, and dampening time along with tolerances. Stating all these values in the same requirement could be seen to violate our combiner and multiple thought rules. Having a requirement that states: “Upon turn on, the system shall supply the initial voltage as shown in drawing xxxxx.” It is much simpler for the drawing to show the magnitude, rise time, allowable overshoot, and dampening time. Stating each as a separate requirement isn’t applicable because all four conditions are part of the one action.

Examples:
Unacceptable: The control system shall close Valves A and B within 5 seconds of the temperature exceeding 95 °C and within 2 seconds of each other. *{This is unacceptable because of the confusing set of conditions. In this case what is meant will be clear if expressed in the form of a diagram.}* *Acceptable*: When the Product temperature exceeds 95 °C, the Control_System shall close Input_Valves as specified in timing diagram 6. *{Note that this assumes, of course, that the timing diagram itself is not ambiguous. If it is not possible to remove the ambiguity through the use of a diagram, it may be better to split the requirement into two.}* *Note: for the above example, if this complex behavior is the result of a design decision, then the requirement needs to be communicated below the line as a design output. The design input requirement should focus on why this behavior is needed.*

Characteristics that are established by this rule:
C3 – Unambiguous C4 - Complete *C5 - Singular*

4.5 COMPLETENESS

4.5.1 R24 - /COMPLETENESS/AVOIDPRONOUNS

Avoid the use of pronouns and indefinite pronouns.

Elaboration:
Repeat nouns in full instead of using pronouns to refer to nouns in other need or requirement statements. Pronouns are words such as “it”, “this”, “that”, “he”, “she”, “they”, and “them.” When writing stories, pronouns are a useful device for avoiding the repetition of nouns; but when writing need and requirement statements, pronouns are effectively cross-references to nouns in other need or requirement statements and, as such, are ambiguous and should be avoided. When originally written, the noun that defines the pronoun may have preceded the pronoun in the previous need or requirement statement, however, as the set of requirements mature, individual requirements may be deleted, reordered, or regrouped, and the defining requirement is no longer nearby. This is especially true when requirements are stored in a requirement management tool where they exist as single statements in a database that may not be in order. To avoid these problems, repeat the proper nouns rather than using a pronoun. Indefinite pronouns are words such as “all”, “another”, “any”, “anybody”, “anything”, “both”, “each”, “either”, “every”, “everybody”, “everyone”, “everything”, “few”, “many”, “most”, “much”, “neither”, “no one”, “nobody”, “none”, “one”, “several”, “some”, “somebody”, “someone”, “something”, and “such.” Indefinite pronouns stand in for unnamed people or things, which makes their meaning subject to interpretation, ambiguous, and unverifiable. See also R32 and R36.

Examples:
Unacceptable: The controller shall send the driver his itinerary for the day. It shall be delivered at least 8 hours prior to his Shift. *{This is unacceptable because the requirement is expressed as two sentences (R19), and the second sentence uses the pronouns "it" and "his."}* *Acceptable*: The Controller shall send the Driver_Itinerary for the day to the Driver less than 8 hours prior to the Driver_Shift. *{Note use of the glossary to define terms and to be explicit about the relationship between the driver, shift, and the itinerary for that particular driver.}*

Characteristics that are established by this rule:
C3 - Unambiguous C4 - Complete C7 - Verifiable

4.5.2 R25 - /COMPLETENESS/USEOFHEADINGS

Avoid relying on headings to support explanation or understanding of the requirement.

Elaboration:
It is a common mistake in document-centric documentation of needs and requirements to use the heading for a specific topic or subject under which the requirement applies to contribute to the explanation of the need or requirement statement. The need or requirement statement should be complete in and of itself and not require the heading for the intent of the need or requirement to be clearly understood. This issue occurs less frequently when using a requirements management tool (RMT) or other SE tool.

Examples:
Example heading: Alert Buzzer Requirements *Unacceptable:* The system shall sound it for greater than 20 minutes. *{This is unacceptable because the requirement uses the pronoun "it" (R24), which requires the heading to understand what "it" means.}* *Acceptable:* The system shall sound the Alert_Buzzer for greater than 20 minutes.

Exceptions and relationships:
The use of headings is acceptable when using the headings to indicate groupings of related requirements or types of requirements. See R40 and R41.

Characteristics that are established by this rule:
C4 - Complete

4.6 REALISM

4.6.1 R26 - /REALISM/AVOIDABSOLUTES

Avoid using unachievable absolutes.

Elaboration:
An absolute, such as "100% availability", is not achievable. Think ahead to verification and validation: how would you prove 100% availability? Even if you could build such a system, could you afford to verify or validate it?

Other examples to avoid are “all”, “always”, and “never” since such absolutes are impossible to verify without an infinite number of verification or validation activities.

Examples:

Unacceptable: The system shall have 100% availability. *{This is unacceptable because 100% is an absolute that is impossible to achieve and verify. Also, available over what time period? }*

Acceptable: The system shall have an Availability of greater than or equal to 98% during operating hours.

{Note use of the glossary to define terms.}

Characteristics that are established by this rule:

C6 – Feasible (Individual requirement)

C7 - Verifiable

C12 – Feasible (Set of requirements)

4.7 CONDITIONS

4.7.1 R27 - /CONDITIONS/EXPLICIT

State applicability conditions explicitly.

Elaboration:

State applicability conditions explicitly instead of leaving applicability to be inferred from the context.

Sometimes need or requirement statements are only applicable under certain conditions. If so, the condition should be repeated in the text of each need or requirement statement, rather than stating the condition and then listing the actions to be taken.

See also R18.

Examples:

Unacceptable: In the event of a fire:

- Power to electromagnetic magnetic fire door catches shall be turned off.
- Security entrances shall be set to Free_Access_Mode.
- Fire escape doors shall be unlocked.

{This is unacceptable because the condition “in the event of a fire” is stated following a list of actions to be taken. Also, note the actions are written in passive voice which violates R2.}

Acceptable:

- In the event of a fire, the Security_System shall turn off power to electromagnetic magnetic fire door catches.
- In the event of a fire, the Security_System shall set security entrances to the Free_Access_Mode.
- In the event of a fire, the Security_System shall unlock fire escape doors.

Characteristics that are established by this rule:

C4 - Complete

C7 - Verifiable

4.7.2 R28 - /CONDITIONS/EXPLICITLISTS

Express the propositional nature of a condition explicitly for a single action instead of giving lists of actions for a specific condition.

Elaboration:
When a list of conditions is given for a single action in a single need or requirement statement, it may not be clear whether all the conditions must hold (a conjunction) or any one of them (a disjunction) for the action to take place. The wording of the need or requirement should make this clear. When the combination of conditions and actions that trigger another action is complex, consideration should be given to the use of a diagram or table (see R23).

Examples:
Unacceptable: The Audit _Clerk shall be able to change the status of the action item when: - the Audit _Clerk originated the item; - the Audit _Clerk is the actionee; and - the Audit _Clerk is the reviewer. *{This is unacceptable because it is not clear whether all the conditions must hold (a conjunction) or any one of them (a disjunction). Also, the requirement contains the phrase "be able to" which violates R11.}* *Acceptable if interpreted as a disjunction*: The Audit_System shall allow the Audit_Clerk to change the status of the action item when one or more of the following conditions hold: - the Audit_Clerk originated the item; - the Audit_Clerk is the actionee; - the Audit_Clerk is the reviewer. *Acceptable if interpreted as a conjunction*: The Audit_System shall allow the Audit_Clerk to change the status of the action item when the following conditions are simultaneously true: - the Audit_Clerk originated the item; and - the Audit_Clerk is the actionee; and - the Audit_Clerk is the reviewer.

Characteristics that are established by this rule:
C3 - Unambiguous
C7 - Verifiable

4.8 UNIQUENESS

4.8.1 R29 - /UNIQUENESS/CLASSIFY

Classify the need or requirement according to the aspects of the problem or system it addresses.

Elaboration:
By classifying needs or requirements by type or category, it is possible to regroup or sort requirements to help identify potential duplications and conflicts within a given classification. The ability to view specific groups of needs or requirements can also assist in identifying what needs or requirements may be missing helping to address the characteristic of completeness as associated with sets of needs or requirements.

It is useful to attach to each requirement statement an attribute associated with the type or category of requirement. (See Section 5, Attributes, A37). Example of such types or categories include: function, performance, operational, interfaces, quality (-ilities), constraints, environmental conditions, ergonomic, safety, security, facility, transportability, training, documentation, testing, quality provision, policy and regulatory, compatibility with existing systems, standards and technical policies, growth capacity, installation, retirement, and disposal.

Examples of organizing needs include form, fit, function, quality, and compliance.

The type/category attribute is most useful because it allows the needs and requirements database to be viewed by a large number of designers and stakeholders for a wide range of users. For example, maintainers could review the database by asking to be shown all the maintenance requirements, engineers developing test plans could extract all verification requirements, and so on. To that end then, the organization would choose types or classifications that are useful for the management of the need or requirement set. As discussed in Section 5, Attributes allow reports to be generated for all needs or requirements of a certain type or category, allowing the identification of duplication, conflicts, or absence of requirements.

How an organization organizes needs and requirements should be clearly defined in the organization's process documents and associated templates for organizing sets of needs and requirements.

Examples:

Example classifications: Functional, Performance, Operational, Reliability, Availability, Maintainability, Safety, Security, Design and Construction Standards, Physical Characteristics.

See Attribute A37– Type/Category for more detailed examples.

See Also R40 - /Modularity/RelatedRequirements

Characteristics that are established by this rule:

C10 - Complete
C11 - Consistent
C12 - Feasible

4.8.2 R30 - /UNIQUENESS/EXPRESSONCE

Express each need and requirement once and only once.

Elaboration:

Avoid including the same or equivalent need and requirement more than once, either as a duplicate or in similar form. Exact duplicates are relatively straightforward to identify; finding similar need or requirement statements with slightly different wording is much more difficult but is aided by the consistent use of defined terms and by classification.

Examples:

Exact duplicates can be found by matching of text strings. The main problem is to identify similarities with different expressions yet are equivalent. For example: "The system shall generate a report of financial transactions containing the information defined in <some standard or contract deliverable listing>" and "The system shall generate a financial report" are overlapping in that the first is a subset of the second.

Avoidance of duplication can be aided by classification so a subset of needs or requirements can be compared.

Characteristics that are established by this rule:
C1 - Necessary
C9 - Conforming
C11 - Consistent
C12 - Feasible

4.9 ABSTRACTION

4.9.1 R31 - /ABSTRACTION/SOLUTIONFREE

Design inputs avoid stating a solution unless there is rationale for constraining the design.

Elaboration:
Design inputs focus on the problem "what" rather than the solution "how."

Needs are communicated at a higher level of abstraction than the requirements that implement the needs. The needs are written from the perspective of the stakeholder expectations rather from the perspective of the system that will result in those expectations being met.

As design inputs, every system endeavor should have a level of needs and requirements that captures the problem to be solved (design inputs) without including solutions (design outputs). System level needs and requirements should provide a system-level requirement for the overall problem to be addressed by design. The first level of architecture may be laid out, but subsystems are considered as black-boxes to be elaborated at the next level down.

When reviewing a requirement that states a solution, again ask "for what purpose? The answer will reveal the real requirement. (Notice that this question is subtly different from simply asking "Why?" and encourages a teleological rather than causal response.) Understanding the concepts of design inputs versus design outputs allows you to ask the question: "Is this requirement addressing what the system needs to do versus how the system needs to do it"? The answer to these questions has the potential to help do three things:

1. Rephrase the requirement in terms of the problem being solved.
2. Determine if the requirement is at the right level.
3. Identify which design input requirement(s) or need(s) the design output requirement is addressing or whether or not the design input requirement is missing.

Often when rationale is provided with a requirement, for requirements that state a solution (design output), the rationale should answer the "for what purpose?" question, and thus the real (and often missing) design input requirement may be extracted from the rationale.

Examples:
General: For a medical diagnostic system

- Stakeholder need statement: "The stakeholders need the diagnostic system to measure [something] with an accuracy as good as or better than similar devices in the market." *[This is an appropriate level of abstraction for a stakeholder need statement, clearly stating the expectation the stakeholders have concerning accuracy, however this would not be a good requirement.]*
- Requirement transformed from the stakeholder need statement: "The diagnostic system shall measure [something] with an accuracy of [xxxxx]." *[The developers have explored various concepts for meeting the need for accuracy, have examined candidate technologies, have*

accessed their maturity (technology readiness level (TRL), and have decided that the value [xxxxxxx] is feasible with acceptable risk for this life cycle stage. As stated, this is a design input requirement.]

- This system level design input accuracy requirement is then allocated to the key parts of the system that have a role in meeting the overall system accuracy requirement. For a medical diagnostic system this could include allocations to the hardware (instrument), assay (biological sample), and software. These allocations could then be further sub-allocated within the hardware, assay, and software entities. As long as the requirements are written on the accuracy allocation and not how that accuracy will be obtained by one of the architectural entities, the requirements are design input requirements. As soon as specific hardware components are named (laser, LEDs emitting light at a specific wavelength, optical system components, specific magnifications, algorithms, formulations, etc,) then the requirements are reflecting design outputs and need to be communicated within deign output artifacts (specifications).

Unacceptable: Traffic lights shall be used to control traffic at the junction. *{This is unacceptable because "Traffic lights" are a solution (design output). This requirement is also written in passive voice – see R2}*

Acceptable (several requirements):

The Traffic_Control_System shall signal "Walk" when the Pedestrian is permitted to cross the road at the junction. *{This is the motivating purpose.}*

The Traffic_Control_System shall limit the Wait_Time of Vehicles traversing the Junction to less than 2 minutes during normal daytime traffic conditions.

Unacceptable: By pressing a button on the traffic-light pillar, the Pedestrian signals his presence, and the light turns red for the traffic to stop. *{This is unacceptable because this requirement contains solution-biased detail – design output. In addition, the requirement is unacceptable because the subject is the user over which we have no control—that is, we cannot tell the user what to do in a requirement statement.}*

Acceptable: The Traffic_Control_System shall allow the Pedestrian to signal intent to cross the road. *{This requirement allows freedom in determining the best solution (design input), which may be a means of automatic detection rather than button pushing.}*

{Note that glossary definitions should be used for "Pedestrian", "Vehicle", and "Junction."}

Exceptions and relationships:

Sometimes solutions have to be described in requirements, even if it is very detailed for a given level, for example if the airworthiness authorities require the use of a specific template for a certain report; or if a naval customer requires that the new naval vessel be equipped with a specific weapon system from a specific supplier; or, if all vehicles in a fleet are required to use the same fuel or the next model make use of a particular engine. In these cases, it is not a premature solution, but a real stakeholder or customer need concerning a constraint. As such this is acceptable as a design input.

However, as a general rule, if something very detailed, design solution specific is expressed as a design input without proper justification, it may be a premature solution and should be communicated as a design output and an appropriate set of design input needs and associated requirements developed that communicate "For what purpose?" which the design output requirements can be traced to.

Examples of issues concerning design outputs expressed as design inputs include:

1. The project is developing an upgrade to an existing system (brownfield SE). Rather than documenting design input "what" requirements, the project team focuses on known solutions and implementations and documenting design output level requirements as design inputs. This is problematic in that the real "for what purpose?" question is not being addressed and the real design input requirements are not communicated.
2. A similar case exists when procuring a commercial off the shelf (COTS) solution and communicating the requirements for that solution as design input requirements rather than in a design output specification. Again, this is problematic in that the real "for what purpose?" question is not being addressed and the real design input requirements are not communicated.
3. A common issue is when an existing system or COTS exists, yet the project develops and documents as design inputs a detailed design output level set of requirements (specification) for the system rather than a set of design input set of "what" requirements that would guide the selection of an appropriate COTS. This can result in a redundant set of requirements that are not necessary, will have to be verified – adding additional cost to the project, yet not addressing the "for what purpose?" design input requirement set that should drive the selection of the specific COTS.
4. As part of the concept maturation process a prototype was developed to access feasibility. The resulting needs and resulting requirements are based on the prototype and an associated trade study that resulted a specific solution that meets the needs of the stakeholders. Rather than communicating the design input requirements, the design output requirements for the prototype are included in the design input set of requirements while not addressing the "for what purpose?" design input requirement set that would drive the design for the prototype.

Characteristics that are established by this rule:

C2 - Appropriate

4.10 QUANTIFIERS

4.10.1 R32 - /QUANTIFIERS/UNIVERSALS

Use "each" instead of "all", "any", or "both" when universal quantification is intended.

Elaboration:

The use of "all", "both", or "any" is confusing because it is hard to distinguish whether the action happens to the whole set or to each element of the set. "All" can also be hard to verify unless "all" can be clearly defined as a closed set. In many cases, the word "all" is unnecessary and can be removed, resulting in a less ambiguous need or requirement statement.

Examples:

Unacceptable: The Operation_Logger shall record any *(or all)* warning messages. *{This is unacceptable because of the use of the word "any", which is then made worse by the addition of "(or all)".}*

Acceptable: The Operation_Logger shall record each Warning_Message. *{Note that Warning_Message must be defined so that it is clear that the system only will record each defined Warning Message.}*

Unacceptable: The Record_Subsystem shall display the Names of both of the Line_Items. *{This is unacceptable because of the use of the word "both".}*

Acceptable: The Record_Subsystem shall display the Name of each Line_Item.

Acceptable: The Record_Subsystem shall display each Line_Item_Name.

Characteristics that are established by this rule:
C3 - Unambiguous
C7 - Verifiable
C8 - Correct

4.11 TOLERANCE

4.11.1 R33 - /TOLERANCE/VALUERANGE

Define quantities with a range of values appropriate to the level stated.

Elaboration:
When it comes to defining performance, single-point values are seldom sufficient and are difficult to test. Ask yourself the question: "if the performance was a little less (or more) than this, would I still buy it?" If the answer is yes, then change the need or requirement statement to reflect this.

It also helps to consider the underlying goal: are you trying to minimize, maximize or optimize something? The answer to this question will help determine whether there is an upper bound, lower bound, or both.

State the quantities contained in a need or requirement statement with ranges or limits with a degree of accuracy that is appropriate to the level at which the need or requirement is stated.

Care should be taken to avoid unspecified value ranges and ensure the quantities are expressed with tolerances or limits. There are two reasons for this:

1) Several requirements may have to be traded against each other, and providing tolerances or limits is a way of describing the trade-space. Seldom is a quantity absolute. A range of values is usually acceptable, providing different performance levels.
2) Verification against a single absolute value is usually not possible or at best very expensive and time consuming, whereas verification against a defined range of values with upper and lower limits makes verification more manageable.

Care should also be taken to ensure the tolerances are no tighter than needed. Tighter tolerances can drive costs both in system design and manufacturing as well as the costs in verifying the system can perform within the tighter tolerances.

Examples:
Unacceptable: The Pumping_Station shall maintain the flow of water at 120 liters per second for 30 minutes. *{This is unacceptable because we don't know whether a solution that carries more or less than the specified quantities is acceptable.}*

Acceptable: The Pumping_Station shall maintain a flow of water at 120 ±10 liters per second for greater than 30 minutes. *{Now we know the range of acceptable flow performance, and that the 30 minutes is a minimum acceptable performance.}*

Unacceptable: The Flight_Information_System shall display the current altitude to approximately 1 meter resolution. *{This is unacceptable because it is imprecise. What is "approximately" in the context of a distance of 1 meter? Who has the option of deciding what is "approximately"? How will "approximately" be verified? What is the acceptable tolerance? Further, altitude is more commonly described in units of feet, not meters, so is +3 feet an acceptable equivalent to 1 meter since 3 feet is less than 1 meter? Note that care must be taken to confirm both units and*

transformation from one set of units to another set to ensure accuracy, acceptability and consistency. See also R6}

Acceptable: The Flight_Information_System shall display Current_Altitude with an accuracy of ±3 feet. *{Note that "Current_Altitude" must be defined in the glossary since there are a number of possible interpretations of the term.}*

Unacceptable: The System shall limit arsenic contamination in the drinking water to allowable levels. Rationale: Arsenic contamination in drinking water can cause health problems. *{While "allowable" is acceptable in a need statement, it is unacceptable in a requirement statement because allowable is ambiguous - allowable by whom? What specific concentration is allowable? In what market?}*

Unacceptable: The System shall limit arsenic contamination in the drinking water to 1 part per trillion. Rationale: Arsenic contamination in drinking water can cause health problems. *{This is unacceptable because the EPA contamination limit in drinking water is 10 parts per billion. Requiring a tighter limit may be beyond the ability of current technology to measure or if measuring concentrations of 1 part per trillion are possible, the cost to do so may be unacceptably high. Also, no range is specified. Using less than is probably the real intent.}*

Acceptable: The System shall limit arsenic contamination in the drinking water to less than10 parts per billion.
Rationale: EPA set the arsenic standard for drinking water at 10 ppb (or 0.010 parts per million). The EPA has determined that concentrations of this level or less will protect consumers from the effects of long-term, chronic exposure to arsenic.

Characteristics that are established by this rule:
C3 - Unambiguous C4 - Complete C6 - Feasible C7 - Verifiable C8 - Correct C12 - Feasible

4.12 QUANTIFICATION

4.12.1 R34 - /QUANTIFICATION/MEASURABLE

Provide specific measurable performance targets appropriate to the level at which the need or requirement is stated.

Elaboration:
Some words signal unmeasured quantification, such as "prompt", "fast", "routine", "maximum", "minimum", "optimum", "nominal", "easy to use", "close quickly", "high speed", "medium-sized", "best practices", and "user-friendly." These are ambiguous and need to be replaced by specific quantities that can be measured.

Examples:
Unacceptable: The system shall use minimum power. *{This is unacceptable because "minimum" is ambiguous.}* *Acceptable:* The system shall use less than or equal to 50W of main power. *{This both takes into account the underlying goal - to minimize power consumption - and provides a measurable target.}*

Unacceptable: The engine shall achieve an emissions level that is at least 5% less than the competition's emission levels 2 years from now. *{This is an actual requirement from marketing to an engineering department. The statement sets a completely unmeasurable end state.}*

Acceptable: The Engine shall achieve an emissions level that is at least xxx. *{where xxx represents the required threshold value.}*

Exceptions and relationships:

Some quantification such as "minimum", "maximum", "optimal" is almost always ambiguous. Other terms may be ambiguous at lower levels but sufficient at the higher levels and as need statements. For example, it may be appropriate that the business state that "The Aircraft shall provide class-leading comfort."—while such a requirement is not quantifiable and therefore not measurable, it may be sufficient for the business to communicate its intentions as a need statement to developers who can then turn comfort into such measurable quantities such as seat dimensions and leg length.

This exception also applies to needs stated at the system or system element levels.

Characteristics that are established by this rule:

C3 - Unambiguous
C4 - Complete
C7 - Verifiable
C12 - Feasible

4.12.2 R35 - /QUANTIFICATION/TEMPORALINDEFINITE

Define temporal dependencies explicitly instead of using indefinite temporal keywords.

Elaboration:

Some words and phrases signal non-specific timing, such as "eventually", "until", "before", "when", "after", "as", "once", "earliest", "latest", "instantaneous", "simultaneous", "while", and "at last", are ambiguous and not verifiable. Indefinite temporal keywords can cause confusion or unintended meaning. These words should be replaced by specific timing constraints.

Examples:

Unacceptable: Continual operation of the pump shall eventually result in the tank being empty. *{This is unacceptable because "eventually" is ambiguous. Also, this statement is not written in the proper form. It is written on "operation" rather than on a requirement on the pump which violates R3.}*

Acceptable: The Pump shall remove greater than 99% of the Fluid from the Tank in less than 3 days of continuous operation.

Characteristics that are established by this rule:

C3 - Unambiguous
C4 - Complete
C7 - Verifiable

4.13 UNIFORMITY OF LANGUAGE

4.13.1 R36 - /UNIFORMLANGUAGE/USECONSISTENTTERMS

Use each term and units of measure consistently throughout need and requirement sets.

Elaboration:

R4 requires the definition of terms in each requirement and R6 requires units of measure to be included with numbers. In addition to those rules, terms and units of measure must be used consistently throughout not only the sets of needs and requirements, but all artifacts developed across all life cycle stages. A common ontology therefore needs to be defined for each project defining terms and units of measure across all artifacts, including the sets of needs and requirements as well as all design output artifacts. Synonyms are not acceptable.

A glossary is very useful to define words precisely. Glossary terms are capitalized to signify that the word or term being used has a specific meaning in the context of the set of statements. Ideally, the glossary used for the sets of needs and requirements is a subset of the project glossary.

Examples:

Unacceptable: It would not be acceptable for one requirement to refer to an entity using one term and another to refer to the same entity using another term.

For example, in a subsystem specification, the following three requirement statements:

The radio shall ....
The receiver shall ....
The terminal shall ....

Or:

The bleed valve shall ....
The high-pressure bleed valve shall ....
The HPBV shall ....

If each term refers to the same subject, the statements need to be modified to use the same word (or, if they are meant to be different, the words must be defined to be so).

Acceptable: Settle on only one term, define it in the glossary, and then use it consistently in each need, requirement, and design output artifact.

Unacceptable: When the Input_Valve is connected to the Water_Source, the Control_Subsystem shall open the Inlet_Valve. *{Two terms are used for the same thing "Input_Valve" and "Inlet_Valve".}*

Acceptable: When the Inlet_Valve is connected to the Water_Source, the Control_Subsystem shall open the Inlet_Valve.

Unacceptable: It would not be acceptable for one requirement to use one unit of measure (e.g. US - feet) and another to use another unit of measure (e.g. Metric - meter).

Characteristics that are established by this rule:

C3 - Unambiguous
C8 - Correct
C9 - Conforming
C11 - Consistent
C13 - Comprehensible
C14 - Able to be Validated

4.13.2 R37 - /UNIFORMLANGUAGE/DEFINEACRONYMS

If acronyms are used in need and requirement statements, use a consistent set of acronyms.

Elaboration:

The same acronym must be used in each need and requirement; various versions of the acronym are not acceptable. The use of different acronyms implies that the two items being referred to are different. Inconsistency in the use of acronyms can lead to ambiguity.

In paper-based specifications, a common rule is to use the full term and the abbreviation or acronym (in brackets) the first time and then use just the abbreviation or acronym from then on.

In requirement sets that are generated from databases, there is no guarantee that the set will be extracted in any particular order, so the old practice is not useful. Acronyms must be used consistently throughout not only the sets of needs and requirements, but all artifacts developed across all life cycle stages. To address this issue, make sure acronyms are defined in an acronym list or glossary.

Acronyms must be written in a consistent way in terms of capitalization and periods. For example, always “CMDP” and not “C.M.D.P.” nor “CmdP”.

Examples:

Unacceptable: It would not be acceptable for one requirement to use the acronym “CP” for command post and another acronym “CMDP” to also refer to the “command post.” The use of two different acronyms implies that the two system elements being referred to are different.

Acceptable: Settle on only one acronym, define it in the list of acronyms, and then use it consistently throughout the requirement set.

Unacceptable: It would not be acceptable for one requirement to refer to the “Global Positioning System” and the remaining requirements refer just to “GPS.”

Acceptable: Use the full term every time or, perhaps more usefully, define the acronym in the acronym list or glossary and use it every time.

Characteristics that are established by this rule:

C3 - Unambiguous
C9 - Conforming
C11 - Consistent
C13 - Comprehensible
C14 - Able to be Validated

4.13.3 R38 - /UNIFORMLANGUAGE/AVOIDABBREVIATIONS

Avoid the use of abbreviations in needs and requirement statements.

Elaboration:

The use of abbreviations adds ambiguity and is to be avoided.

Examples:

Unacceptable: It would not be acceptable for one requirement to refer to the abbreviation “op” meaning operation and another to refer to the “op” meaning the operator.

Acceptable: Avoid the abbreviation and use the full term every time.

Characteristics that are established by this rule:

C9 - Conforming
C11 - Consistent

C13 - Comprehensible
C14 - Able to be Validated

4.13.4 R39 - /UniformLanguage/StyleGuide

Use a project-wide style guide for individual need and requirement statements.

Elaboration:

The style guide provides a template and patterns for developing requirements, defines the attributes that the organization chooses to document with each requirement statement, selects the requirement patterns to be used for specific types of requirements, and defines what other information needs to be included (such as the glossary).

The style guide should also list the rules the organization wants to use (based on this guide). When managing requirements electronically (versus in a printed document) within a requirement management tool (RMT) or other systems engineering tool, this information is the basis of the schema that defines the organization of the data in the tool database.

To ensure need and requirement statements are consistently structured, the organization needs to include pre-defined patterns in their development and management process. The patterns can come from an international standard or an organizational standard set of patterns based on type of stakeholder need or requirement. If using a requirement management tool, a standard schema should be defined as part of the requirement development and management process. The patterns need to be tailored to the organization's domain and the different types of products within that domain.

Patterns for individual need and requirement statements help ensure consistency and completeness of the individual statements (see Appendix B).

See also R1.

Examples:

For example, each organization should have a style guide or schema to address such issues as the selection and definition of what need and requirement patterns should be used for writing needs and requirements of a particular type, the selection and use of attributes, standard abbreviations and acronyms, layout and use of figures and tables, layout of needs and requirements documents or databases and, need and requirement statement numbering.

Terms used throughout the sets of needs and requirements are defined in an ontology or at least a project glossary.

Characteristics that are established by this rule:

C5 - Singular
C9 - Conforming
C11 - Consistent
C13 - Comprehensible
C14 - Able to be Validated

4.14 MODULARITY

4.14.1 R40 - /MODULARITY/RELATEDREQUIREMENTS

Group related requirements together.

Elaboration:
Need and requirements statements that belong together should be grouped together. This is a good principle for structuring a requirements document or organizing requirements with a set documented within an RMT or another SE tool. One approach is to put requirements into groups based on their classification. See also R29, R41, and A37. This will often mean that requirements are grouped by feature, bringing a functional requirement together with a number of related constraints and performance requirements and verification requirements. An example grouping could be: form, fit, function, quality, and compliance. This grouping forces the team to look at the system from each of these perspectives helping to ensure completeness of the sets. Methodologies that only focus on function and fit will result in an incomplete set of needs and requirements. In functional decomposition, functional requirements are grouped by the function that originates them. Performance requirements are grouped with the functions they apply to. This grouping helps to ensure completeness of the sets and makes it easier to identify overlapping or redundant requirements. The template document structure may include headings to help organize requirements into logical groups. See R41.

Examples:
Requirements may be related by: *- type (for example, safety requirements);* *- scenario (for example, requirements arising from a single scenario).* *See R29 and A37 for more examples*

Characteristics that are established by this rule:
C9 - Conforming C11 - Consistent C13 - Comprehensible

4.14.2 R41 - /MODULARITY/STRUCTURED

Conform to a defined structure or template for sets of needs and requirements.

Elaboration:
A well-structured set of needs and requirements allows an understanding of the whole set to be assimilated without undue cognitive loading on the reader. Despite what has been stated about the completeness of individual need and requirement statements, it is often easier to understand when it is placed in context with other related requirements.

Whether presented in document structure or as the result of database queries, the ability to draw together related groups of needs or requirements is a vital tool in identifying repetition and conflict between need or requirement statements.

Templates for organizing needs and requirements will help ensure consistency and completeness of your requirement set.

Examples:

For sets of needs or requirements, an outline can be defined that organizes them in categories or types such as: functional/performance, operational, quality (-ilities), design and construction standards, and physical characteristics. The outlines can come from international standards or an organizational standard tailored to the organization's domain and different types of products within that domain. (The organization for system level needs and requirements may be different than the organization of a set of software needs and requirements.)

Characteristics that are established by this rule

C10 - Complete
C11 - Consistent
C13 - Comprehensible
C14 - Able to be Validated

5 ATTRIBUTES OF NEED AND REQUIREMENT STATEMENTS

In this section a list of attributes is provided that can be associated with each need and requirement statement. The need or requirement statement plus its attributes results in a need or requirement expression as defined in section 1 (Wheatcraft, et al, 2015).

Note: Many of the attributes listed are useful for both managing needs as well as requirements. Others may be more useful as applied to only requirements.

This list is not exhaustive and is not meant to be prescriptive or proscriptive in any way. It is not the intention that an organization should include all of these attributes when defining needs or requirement expressions. The purpose is to present a set of attributes that have been, and are being used, by various organizations. An organization may select a reduced set of these attributes or may well have other attributes that apply to the organization's unique domain and product line.

The important thing is that organizations need to define an attribute scheme for their project. This attribute scheme will be specific to the domain, project, and company. If using a requirement management tool (RMT) or other SE tool, the attributes the project chooses needs to be implemented in the tool schema at the beginning of the project.

Although it is unlikely an organization would include all the attributes listed, there is a subset of these attributes that is proposed to represent a minimum set that the authors recommend be defined for each need or requirement. This minimum set supports the mandatory characteristics of well-formed need and requirement statements, and also supports the maintenance of the need and requirement statements, especially configuration management and tracking the progress of the system design and verification activities throughout the product development life cycle. In the discussion below, the attributes in that minimum set are annotated with an asterisk ("*").

The listed attributes are organized within the following four broad categories:

- *Attributes to help define the need or requirement and its intent.*
- *Attributes associated with the SOI and its verification or validation.*
- *Attributes to help manage the needs or requirements.*
- *Attributes to show applicability and enable reuse of the needs and requirements.*

Note: In the following definitions, the term SOI is used. The SOI is the entity, system, or system element to which the need or requirement statement applies. The SOI could exist at any level. SOI is closely associated with the characteristic of a well-formed need or requirement – C2-Appropriate [to the level].

5.1 ATTRIBUTES TO HELP DEFINE THE NEED OR REQUIREMENT AND ITS INTENT

This set of attributes is used by both the author of the need or requirement and anyone that will be reviewing the need or requirement, implementing the need or requirement, or involved in the various design, verification and validation activities throughout the system development life cycle. These attributes help ensure the need or requirement statements have the characteristics of well-formed need or requirement statements defined in Section 2 as well as help understand the intent and reason for the statement.

5.1.1 A1 – RATIONALE*

Rationale states the reason for the need or requirement statement's existence. Rationale defines why the need or requirement is needed and other information relevant to better understand the reason for and intent of the need or requirement. Rationale can also record assumptions that were made when writing the need or requirement, the source of numbers, and what design effort drove the requirement. Rationale, along with the attributes of A4 - *Trace to Parent* and A5 - *Trace to Source*, helps to support the claim that the requirement characteristic C1 - *Necessary* is true.

5.1.2 A2 – SOI PRIMARY VERIFICATION OR VALIDATION METHOD*

The primary verification or validation method for each need or requirement states the planned primary method of verification or validation (test, demonstration, inspection, analysis) that is being proposed to provide proof the designed and built system meets the requirement (verification) or stakeholder need (validation). Identifying the SOI primary verification or validation method helps ensure the need or requirement characteristics C3- *Unambiguous*, C7 - *Verifiable*, and C8 - *Correct* are true. This attribute can be used to build an SOI verification or validation matrix. *This attribute may also be classified under attributes associated with SOI verification or validation defined in Section 5.2.*

5.1.3 A3 – SOI VERIFICATION OR VALIDATION APPROACH

Approach or strategy suggested to verify the SOI meets a requirement or validate the SOI meets a need statement per the primary verification or validation method. For example, for a requirement "The system shall do X.", the primary method may be Test. Then the verification approach or activity may be "The Test will consist of …." with a definition of what the test comprises. Other information may also be included: "The ability of the system to do X will be proven when the Test shows …" or "Verification will be considered successful when the Test shows …" This is a definition of the success criteria for verifying the system does meet the system requirement or validation that the system meets the need. Some tools enable the definition of a separate set of "verification requirements" or "validation requirements". Using this approach, this attribute would be a link (trace) to those requirements. Identifying the SOI verification or validation approach helps to ensure the need and requirement characteristics C3 - *Unambiguous*, C7 - *Verifiable*, and C8 - *Correct* are true. This attribute can also be used to build an SOI verification or validation matrix. *This attribute may also be classified under attributes associated with system verification and validation defined in Section 5.2.*

5.1.4 A4 – TRACE TO PARENT *

System level requirements are a transformation from one or more needs, so at the system level all requirements must trace to their parent stakeholder need(s). Requirements at one level are implemented at the next level of the system architecture via allocation (see A8, Allocation). A child requirement is one that has been derived or decomposed from the allocated parent. The achievement of each of the children requirements will lead to the achievement of the parent requirement. Each of the children requirements must be traced to its parent requirement (and then to any antecedent requirement, and ultimately to the system need/mission. When managing requirements in a requirement management tool (RMT) or other SE tool, when a trace is established from the child to the parent, a reverse trace is also established between the parent and that child requirement. This is commonly referred to as "bi-directional" traceability. Having this knowledge allows the SE to ensure all parents have children, each of the children are necessary and sufficient to meet the intent of the parents, all parents have the correct children, all children requirements have at least one parent, and all children requirements trace to the correct parents. Traceability between

parents and children is also key to assessing the impact of changes, no matter which level of the architecture the change occurs.

For those using an RMT or other SE tool, the tool should support the concept of traceability and a separate attribute may not be needed. However, for those are using common office applications to document their requirements, trace to parent is accomplished via this attribute. For a large set of requirements maintaining these traces within an common office application can be very difficult and time consuming – a major reason organizations adopt the use of an RMT or other SE tool!

Given the right side of the "Vee" shown in Figure 5 is a bottom up process, verification that the children requirements have been met should occur prior to verification that the parent requirement has been met. Successful completion of verification activities showing that the children requirements have been met should be a prerequisite to verifying the parent has been met. The attribute a4 - *Trace to Parent*, along with a1 - *Rationale* and A5 - *Trace to Source*, helps to ensure the requirement characteristic of C1 - *Necessary* is met. *This attribute may also be classified under attributes used to manage the requirement.*

5.1.5 A5 – TRACE TO SOURCE*

Each need and requirement must be able to be traced to its source. For requirements, this is different from tracing to a parent requirement because it identifies where the requirement came from and/or how the requirement content was determined (rather which specific requirement is its parent). Each need must trace to a source to establish where the need was derived from. Examples of sources include mission need, goals, or objectives, constraints, system concepts, user stories, use cases, models, analysis, documents, regulations, standards, risks, derived from a trade study, interviews with a stakeholder, a specific stakeholder, minutes of stakeholder workshop, or Engineering Change Proposal (ECP). Sources could also be a functional area within an enterprise or business unit (marketing, safety, compliance, quality, engineering, manufacturing, etc.).

Maintaining this trace is key to being able to show compliance to higher level organizational requirements, customer requirements, regulations, and standards. This trace is also important in terms of change management. If something related to a source changes, then the need or requirement traced to that source may also need to be changed. For example, if the analysis used to generate a number in a requirement is updated as the design matures, the requirement must also be updated. A5 - *Trace to Source*, along with A1 - *Rationale* and A4 - *Trace to Parent*, helps to ensure the requirement characteristic of C1 - *Necessary* is met.

5.1.6 A6 – CONDITION OF USE

Description of the operational conditions of use expected in which the need or requirement applies. Some organizations prefer to include the A6 - *Condition of Use* as part of the requirement statement (see R18); others state the conditions of use as separate requirements to define the operation environment in which the system will be used. Understanding the condition of use is key to a proper implementation in design as well as system verification and validation activities. *This attribute may also be classified under attributes associated with verification and system validation defined in Section 5.2.*

5.1.7 A7 – STATES AND MODES

The state or mode of the system to which the requirement applies. Some systems have various states and modes, each having a different set of requirements that apply to the

specific state or mode. If the system is structured in this way, this attribute allows requirements to be assigned to the applicable states and modes.

5.1.8 A8 – ALLOCATION

Requirements at one level are allocated to parts of the architecture (functional or physical depending on the level) for implementation. There are two types of allocation. The first type, allocation of resources, involves allocating a quantity at one level to the next level, where the resource implementation is shared between entities at the next level (such as performance, power, mass, reliability, accuracy, or precision). The second type of allocation is allocation of responsibility where requirements at one level are assigned to entities at the next level for implementation.

For blank sheet or green field systems, when allocation is performed there are no requirements at the next level, so allocation is linking a requirement at one level to entities (parts of the architecture or organizational units) at the next level. At the level the requirements are allocated to, each entity will develop children requirements via either decomposition or derivation that will result, collectively, in the intent of the parent allocated requirement to be met. These children requirements are then traced back to their parent (see A4 - *Trace to Parent*.) These children requirements are also dependent—a change to one would result in a change in one or more of the other children requirements. Because of this dependency, all children having a common parent would need to also be linked together. (See A30 - *Trace to Peer*.)

For those using an RMT or other SE tool, the tool should support the concept of allocation and a separate attribute may not be needed. However, for those are using common office applications to document their requirements, allocation is accomplished via this attribute. For a large set of requirements maintaining these allocations and resulting traces within an common office application can be very difficult and time consuming – a major reason organizations adopt the use of an RMT or other SE tool!

5.2 ATTRIBUTES ASSOCIATED WITH THE SOI AND ITS VERIFICATION OR VALIDATION

This set of attributes, along with A2 - *SOI Primary Verification or Validation Method*, A3 - *SOI Verification or Validation Approach*, and A6 - *Condition of Use* described above, is used to help track and manage the SOI verification and validation activities and status to make sure the designed and built SOI meets the requirement (verification) or need (validation). This set of attributes can be used for developing a SOI verification or validation matrix as well as provide metrics that management can use to track the status of the project's SOI verification and validation activities.

5.2.1 A9 – SOI VERIFICATION OR VALIDATION LEVEL

Architecture level at which it will be proven that the SOI meets the requirement (verification) or need (validation). Possible values could include: <system>, <subsystem>, <assembly>, <component>, <hardware>, <software>, <integration>, etc. The actual name of the level can be organization/domain specific based on the product breakdown structure or architecture description document.

5.2.2 A10 – SOI VERIFICATION OR VALIDATION PHASE

Life-cycle phase in which the SOI will be proven to meet the requirement (verification) or need (validation). Possible values could include: design, coding/testing, proof testing, qualification, and acceptance. The actual name of the phase can be organization/domain specific, such as software, hardware, and hybrid.

5.2.3 A11 – SOI VERIFICATION OR VALIDATION RESULTS

The results of each SOI verification or validation activity will most often be contained in a separate document or electronic record. This attribute traces each requirement to the associated SOI verification or validation activity results document or record. Possible values could include: complete – successful, complete – unsuccessful, verified (passed) or failed. This attribute allows reports to be generated, for example, percent of requirements whose SOI verification that have been successfully completed or percent of needs whose SOI validation has been successfully completed.

5.2.4 A12 – SOI VERIFICATION OR VALIDATION STATUS

Indicates the status of the SOI verification or validation activities including sign-off/approval of the SOI verification or validation stating whether the system has been verified that it meets the requirement or validated that it meets the need. Possible values could include not started, in work, percent complete, complete, sign-off in work, approved, waived, and deviation approved. This attribute allows reports to be generated, for example, percentage of requirements whose system verification have been approved or needs whose system validation has been accepted by the customer.

5.3 ATTRIBUTES TO HELP MAINTAIN THE REQUIREMENTS

This set of attributes is used by management to maintain the needs and set of needs and requirements and the set of requirements. Areas of prime importance to management, in addition to the above attributes dealing with SOI verification and validation activities, include managing risk, managing change, ensuring consistency and completeness, and having insight into the status of needs and requirements implementation across all life cycle stage activities. Managing risk is a key role of management as well as making sure needs and requirements are tracked if they are high priority or essential to project success. Including attributes dealing with these topics allows reports to be generated by various SE tools that will help management keep the project on track to success and manage changes not only as they relate to individual needs and requirements but also changes to the project such as budget cuts and problems resulting in cost overruns and schedule slips.

5.3.1 A13 – UNIQUE IDENTIFIER*

A unique identifier, which can be either a number or mixture of characters and numbers used to refer to the specific need or requirement. The unique identifier is not a paragraph number. It can be a separate identifier or automatically assigned by whatever RMT or SE tool the organization is using. This identifier is used once and never reused. A unique identifier is also used to link needs and requirements in support of the flow down of requirements (allocation), traceability, and to establish peer-to-peer relationships. Some organizations include in the unique identifier codes that relate to the SOI or level of architecture to which the need or requirement applies e.g. [SOI]SNxxxx. or [SOI]SRxxxx. Organizations should never use a document paragraph number as a requirement unique identifier.

5.3.2 A14 – UNIQUE NAME

A unique name or title for the requirement that reflects the main thought the requirement is addressing. This is useful to provide a short title to allow the requirements to be output as a tree structure or is graphical form. Caution should be used when using this attribute as discussed in R25.

5.3.3 A15 – ORIGINATOR/AUTHOR*

The person submitting the need or requirement and/or the person responsible for entering the need or requirement into the RMT. When entering a need or requirement into an RMT or other SE tool, the tool may automatically log the name of the person entering the requirement.

5.3.4 A16 – DATE REQUIREMENT ENTERED

The date the need or requirement was entered into the RMT or other SE tool. This date may be automatically generated by the tool when the need or requirement is entered into the tool.

5.3.5 A17 – OWNER*

The person or element of the organization that maintains the need or requirement, who has the right to say something about this need or requirement, approves changes to the need or requirement, and reports the status of the need or requirement. The same person could be both the Owner and the Source [of the need or requirement], but it is recommended that Owner and Source are two different attributes. The *Owner* maintains the attributes A26 – *Need or Requirement Verification Status*, A27 - *Need or Requirement Validation Status*, and A28 - *Status (of need or requirement)*.

5.3.6 A18 – STAKEHOLDERS

List of key stakeholders that have a stake in the implementation of the need or requirement and who will be involved in the review and approval of the need or requirement as well as any proposed changes. In RMTs and other SE tools that support collaboration, this list of stakeholders are those that will be automatically notified by the tool when either the need or requirement statement or any associated attribute changes.

5.3.7 A19 – CHANGE BOARD

The organizational entity that has configuration management authority over the need and requirement and has the authority to baseline the need or requirement and approve proposed changes to the need or set of needs or changes to the requirement and set of requirements of which it is a part.

5.3.8 A20 – CHANGE STATUS

An indication of the status of a proposed change. For example: submitted, in review, withdrawn, approved, disapproved.

5.3.9 A21 – VERSION NUMBER

An indication of the version of the need or requirement. This is to make sure that the correct version is being implemented as well as to provide an indication of the volatility of the need or requirement. A need or requirement that has a lot of change could indicate a problem or risk to the project. The version number may be automatically assigned by the RMT or other SE tool being used.

5.3.10 A22 – APPROVAL DATE

Date the current version of the need or requirement was approved. This may be automatically assigned by the RMT or another SE tool.

5.3.11 A23 – Date of Last Change

Date the need or requirement was last changed. This may be automatically assigned by the RMT or another SE tool.

5.3.12 A24 – Stability

An indication of the likelihood that the need or requirement will change. Some needs and requirements, when first proposed, will have numbers that are best guesses at the time, or the actual value is not readily available when the requirement has been entered into the RMT. The number may not be agreed to by all stakeholders or there may be an issue on the achievability of the need or requirement. Some needs or requirements will have a to-be-determined (TBD), to-be-computed (TBC), or to-be-resolved (TBR) in the text. Stability is directly related to a high or medium risk need or requirement (see the risk of implementation attribute below for more information). Possible values could include stable, likely to change, and incomplete.

5.3.13 A25 – Responsible Person

Person responsible for ensuring that the need or requirement is satisfied. This is different from need or requirement owner defined previously. The *Responsible Person* maintains the attribute *Status [of implementation]* - see A29.

5.3.14 A26 – Need or Requirement Verification Status*

An indication that the need has been validated or requirement has been verified. As defined in Section 1.7, need or requirement verification is the process of ensuring the need requirement meets the rules and characteristics (necessary, singular, conforming, appropriate, correct, unambiguous, complete, feasible, and verifiable) for a well-formed need or requirement statement as defined in this Guide or a comparable guide or checklist developed by the organization. Possible values include true/false, yes/no, or not started, in work, complete, and approved.

5.3.15 A27 – Need or Requirement Validation Status*

An indication the need or requirement has been validated. As defined in Section 1.7, requirement validation is the confirmation the requirement is an agreed-to transformation that clearly communicates the needs of the stakeholders in a language understood by the developers. Need validation is the confirmation the need is an agreed-to transformation that clearly communicates the concepts and stakeholder expectations in a language understood by the requirement writers. Need and validation activities need to involve the customers and users and all other stakeholders of the system being developed. Possible values include true/false, yes/no, not started, in work, complete, and approved.

5.3.16 A28 – Status (of the need or requirement)

Maturity of the need or requirement. Possible values include: draft, in development, ready for review, in review and approved. In general, the set of needs or requirements would not be baselined until all the individual need or requirement verification and validation status has been set to indicate that verification and validation of the need or requirement statement is complete and has been approved. A status of “approved” should only be allowed if the above need or requirement verification and validation status are both “True”; the requirement expression is stable; A34 - *Risk (of implementation)* is at an acceptable level and has the organization’s mandatory attributes defined.

5.3.17 A29 – STATUS (OF IMPLEMENTATION)

For a need, an indicator of the status of transformation of a need into a design input requirement(s). Possible values include transformation complete, transformation in work, and no plans to transform the need into requirement(s). When the transformation is complete, all resulting requirements need to trace back to the need(s) from which it was transformed. See also A5 - *Trace to Source*. All needs will have at least one implementing requirement.

For a requirement, an indicator of the status of the implementation or realization of the requirement in the design outputs. Possible values include requirement implemented in design; requirement not implemented and no plan to do so; and requirement not implemented but have plan to do so. If implemented in the design outputs and your SE Tool set allows, this attribute could be a trace between the design outputs and the design input requirement. If using an SE modeling tool, this attribute could indicate the requirement has been implemented in the model or in linked to a portion of the model.

Note: if there is no plan to implement a need or requirement, a change, waiver, or deviation would have to be submitted and approved by the configuration control authority (A19 – Change Board).

5.3.18 A30 – TRACE TO INTERFACE DEFINITION

The interactions between two systems are described in interface definitions, which are often contained in a document (or other similar electronic representation) which has a title such as an Interface Control Document (ICD), an Interface Agreement Document (IAD), an Internal Interface Definition Document (IIDD), an Interface Definition Document (IDD), or an Interface Requirements Document (IRD). In software, the interface definition could be in a data dictionary, an application program interface (API), or software development kit (SDK).

The interface requirements contained in each of the interacting systems' requirement sets will include a reference to where the interaction is defined. This attribute provides a link tracing the interface requirements to where the interaction is defined.

This attribute facilitates reports out of the RMT or other SE tool so that, for any definition, it is obvious which interface requirements invoke that definition. This aids in change control. If any interface requirement changes, the definition may need to be changed. If the definition changes, interface requirements that are linked to that definition may need to be accessed or changed as well (or at least the owners of the interface requirement need to be made aware that the definition changed).

This attribute also enables reports to show completeness (C4). If no requirement points to an interface definition or only one requirement points to the definition, there may be a missing interface requirement.

5.3.19 A31 – TRACE TO PEER REQUIREMENTS

This attribute links requirements that are related to each other (other than parent-child) at the same level. Peer requirements may be related for a number of reasons, such as: they may be co-dependent (a change to one could require a change to the other), or bundled (you can only have this requirement if you also have the one it is linked to), or a complementary interface requirement of another system to which your system has an interface with.

This trace may also result from the allocation process. If a resource was allocated from one level to multiple entities at the next level, the values of the resulting children requirements are dependent – a change to one would result in a change in one or more of the other

children requirements. Because of this dependency, all children having a common parent would need to be linked together. (See A8 - *Allocation*.)

This attribute enables reports to show the characteristics of a set of requirements C10 - *Complete* and C11 - *Consistent*. If there is an interface requirement that indicates the SOI is interacting with another SOI, but that SOI does not have the complementary interface requirement there is probably a missing requirement. For most RMTs or other SE tools, having a trace from one co-dependent requirement to another results in an automatic bi-directional trace such that if there is a proposed change to one, the impact of that change on the other(s) can be accessed.

5.3.20 A32 – PRIORITY*

This is how important the need or requirement is to the stakeholders. It may not be a critical/essential requirement—that is, one the system must possess, or it won't work at all (see A33 - *Priority*)—but may simply something that the stakeholder(s) hold very dear. Priority may be characterized in terms of a number (1,2,3) or label (high, medium, low). (The reason for classifying a need or requirement as high priority should be communicated in the rationale attribute (A1 - *Rationale*). Assuming each child requirement is necessary and sufficient, their priority will be inherited from their parent requirement. In the case of needs, all requirements transformed from that need, would inherit the priority of the stakeholder need.

High priority needs and requirements must always be met for the stakeholder expectations to be met; lower priority needs and requirements may be traded off when conflicts occur or when there are budget or schedule issues resulting in a de-scope effort. If there is a need to trade-off (modify or delete) a need or requirement, a change request must be submitted to the organization having configuration management authority for the set of requirements (A19 – *Change Board*).

5.3.21 A33 – CRITICALITY OR ESSENTIALITY*

A critical or essential need or requirement is one that the system must achieve for the system to function at all—that is, without it, the system will not be able to achieve its primary purpose or intended use. A critical or essential requirement would also be included in the system's key performance requirement set. Criticality/essentiality is most often characterized in terms of yes or no. Not all input needs or requirements are critical or essential. (The reason for classifying a stakeholder need or requirement as critical or essential should be communicated in the rationale attribute (A1 - *Rationale*).

Assuming each child requirement is necessary and sufficient, their criticality or essentiality will be inherited from their parent (requirement or need from which the requirement was transformed). Critical or essential needs and requirements must always be met. Unlike high priority needs or requirements, critical or essential requirements may not be traded off when there are budget or schedule issues. If a critical need or requirement cannot be met, the feasibility of the whole project will have to be re-examined. To prevent this from happening, *the authors advocate that projects identify at least one feasible concept prior to baselining the set of needs*.

While highly critical/essential requirements will also have high priorities, it is possible to have a non-critical yet high priority requirement.

5.3.22 A34 – RISK (OF IMPLEMENTATION)*

A value assigned to each need or requirement indicating the risk of the need or requirement not being met.

Risk of implementation may be characterized in terms of a number (1, 2, 3) or a label (high, medium, low). Some organizations communicate risk as a cross product of likelihood and impact (a high likelihood and a high impact would be represented in a risk matrix as 1×1, while a low likelihood and high impact would be 3×1). Some organizations use a 3×3 risk matrix, others a more granular 5×5 matrix. An unstable requirement is also a risk factor (see A24 - *Stability*).

Risk can also address feasibility/attainability in terms of technology, schedule, cost, politics, etc. If the technology needed to meet the requirement is new with a low maturity (low TRL), the risk is higher than if using a more mature technology that has been used successfully in other similar projects. The requirement can be high risk if the cost and time to develop a technology is outside what has been planned for the project. Risk may also be inherited from a parent requirement.

Needs or requirements that are at risk include those that fail to have the characteristics defined in this Guide. Failing to have these characteristics can result in the need or requirement not being implemented (will fail system verification) and needs not being realized (will fail system validation).

5.3.23 A35 – Risk (Mitigation)

This attribute indicates whether the stakeholder need or requirement is part of a risk mitigation.

As part of scope definition and concept maturation activities, projects need to assess program/project, development, and operational risks. As a result, some risks are accepted (no actions taken other than monitoring) while other risks are mitigated. Those risks to be mitigated can be allocated to people, processes, and/or products (systems under development). For those risks allocated to the system under development, the management of the risk mitigation is needed throughout the develop life cycle stage activities. To do this, a concept for mitigating the risk is defined and the associated stakeholder needs to implement this concept are defined. Requirements are transformed from these needs and allocated to lower levels. Verification requirements are also defined for these requirements.

In order to manage the risk mitigation and to ensure the risks allocated to the system under development have been mitigated, the inclusion of this attribute is needed. This attribute, along with trace to parent, trace to source, allocation, priority, and critically/essentiality, allow the risk mitigation to be managed throughout the development life cycle.

Risk (to be mitigated) may be characterized in terms of a value of yes or no. *Note the actual risk will be communicated as a cross product of likelihood and impact and included in the organizations risk tracking database. Having this attribute will allow the status of the mitigation to be tracked.*

This attribute, along with A4 - *Trace to Parent*, A5 -*Trace to Source*, A8 - *Allocation*, A32 - *Priority*, and A33 - *Critically or Essentiality*, allow the risk mitigation to be managed throughout the development lifecycle.

5.3.24 A36 – Key Driving Need or Requirement (KDN/KDR)

A KDR is a need or requirement that, to implement, can have a large impact on cost or schedule. A KDR can be of any priority or criticality/essentiality. Knowing the impact that a KDN/KDR has on the design allows better management of needs and requirements. If your project is in trouble you may want to consider deleting a low-priority, high-risk KDN/KDR. If there is a need to delete or waive a KDN/KDR need or requirement, a change request must be submitted to the organization having configuration management authority for the set of needs and requirements.

5.3.25 A37 – ADDITIONAL COMMENTS

Generic comment field that can be used to document possible issues with the requirement, any conflicts, status of negotiations, actions, design notes, implementation notes, etc. Further, evaluation and prototyping of the system concept may have identified important guidelines for the implementation of the need or requirement. This information may be useful as the need or requirement is being reviewed and serves as a place to document information not addressed by other attributes.

5.3.26 A38 – TYPE/CATEGORY

Each organization will define types or categories to which a need or requirement fits, based on how they may wish to organize their requirements. The *Type/Category* field is most useful because it allows the database to be viewed by a large number of designers and stakeholders for a wide range of uses. For example, maintainers could review the database by asking to be shown all the maintenance needs and requirements, engineers developing test plans could extract all corrosion-control needs and requirements, and so on. *See also the modularity rules R40 - RelatedRequirements together, and R41 – Structured.*

Examples of *Type/Category* of requirements include:

a. Function: Functional/Performance;
b. Fit: Operational: interactions with external systems - input, output, external interfaces, environmental, facility, ergonomic, compatibility with existing systems, logistics, users, training, installation, transportation, storage;
c. Quality (-ilities): reliability, availability, maintainability, accessibility, safety, security, transportability, quality provisions, growth capacity;
d. Form: physical characteristics;
e. Compliance:
    a. standards and regulations - policy and regulatory;
    b. constraints - imposed on the project and the project must show compliance;
    c. business rules - a rule imposed by the enterprise or business unit;
    d. business requirements - a requirement imposed by the enterprise or business unit.

5.4 ATTRIBUTES TO SHOW APPLICABILITY AND ENABLE REUSE

These attributes may be used by an organization that has a family of similar product lines to identify the applicability of the need or requirement (to product line, region, country, etc.) and reuse of stakeholder need or requirement that may apply to a new product being developed or upgrade of an existing product.

5.4.1 A39 – APPLICABILITY

Can be used to indicate which increment, build, release, or model a need or requirement applies to.

5.4.2 A40 – REGION

Region where the product will be marketed, sold, and used.

5.4.3 A41 – COUNTRY

Country(ies) where the product will be marketed, sold, and used.

5.4.4 A42 – STATE/PROVINCE

State(s) or province(s) within a country or region where the product will be marketed, sold, and used.

5.4.5 A43 – APPLICATION

The applicability to a specific product within a product line.

5.4.6 A44 – MARKET SEGMENT

Segment of the market that will be using the product.

5.4.7 A45 – BUSINESS UNIT

A specific business unit within the enterprise that produces the product.

5.4.8 A46 – BUSINESS (PRODUCT) LINE

A specific brand or product line within a given business unit.

5.5 GUIDANCE FOR USING ATTRIBUTES

The reason for using attributes is to better manage your project. Given that needs and requirements are the common threads that tie each of the systems engineering product development life-cycle process activities together, having insight into these activities is necessary to manage the project effectively.

Hull, Jackson and Dick (2011) describe attributes in terms of how they support the engineering process, stating: “Attributes allow the information associated with a single requirement to be structured for ease of processing, filtering, sorting, etc. Attributes can be used to support many of the abilities [associated with requirements engineering], enabling the requirements to be sorted or selected for further action, and enabling the requirements development process itself to be controlled.”

Wiegers (2003) makes the point that a key advantage of using an RMT is the ability to manage both the requirement and its associated attributes. “The tools let you filter, sort, and query the database to view selected subsets of requirements based on their attribute values.”

A major feature of a good RMT or other SE tool is the reports it can generate. Managers can define specific dash boards and reports that target various aspects of their project based on the attributes. Some RMTs and other SE tools allow you to define a dashboard display that includes graphics based on the selected attributes. This is only possible if the project includes in the RMT or SE tool schema the attributes that contain the information needed to produce these reports and dashboards at the beginning of the project.

Knowing the priority helps to plan better project activities. Knowing the risk allows mitigation of that risk. Combining risk and priority or criticality/essentiality can provide management with valuable information they can use to manage the project’s needs and requirements and associated risks. Identifying needs and requirements that are both high priority (or criticality/essentiality) and high risk, allows management to focus on the most important issues. Again, you can create a 3×3 matrix similar to the risk matrix, but this time one axis is risk and the other priority or criticality/essentiality. If these attributes are not documented for each need and associated requirement(s), how will management know which are high priority or critical/essential AND high risk?

Knowing the criticality/essentiality of a requirement or the fact a need is a KDN or a requirement is a KDR allows managers and systems engineers to plan accordingly. When under schedule or budget pressure, a KDN or KDR that is low priority or low criticality, but high risk, may be a candidate for deletion.

Activities like system verification and system validation are key cost and schedule drivers on any project. Knowing the priority, critically/essentiality, and risk of a need or requirement is valuable information when selecting the SOI verification or validation method and approach. Knowing the proposed method and approach of system verification or system validation facilitates the development of a more realistic budget and schedule. Knowing the status of the requirement and status of the system verification and system validation activities allow management of the budget and schedule so actions can be taken when there are indications of problems that could lead to budget overruns and schedule slips before they become real problems.

A major source of budget and schedule issues is change. Attributes such as unique identifier (A13), unique name (A14), rationale (A1), owner (A17), version number (A21), change status (A20), change board (A19), priority (A32), criticality or essentiality (A33), trace to parent (A4), trace to source (A5), trace to peer requirements (A31), stability (A24), and KDN/KDR (A36) allow managers to access the change and the potential impacts of that change.

Attributes that link artifacts from one system engineering life cycle to another (need to requirements (A5), parent to child (A4), peer-to-peer (A31), requirements to design outputs (A8), requirements to system verification (A9) and system validation activities (A10)) aid greatly in change management and accessing the impacts of change across all life cycle activities.

A word of caution; as with the use of all information, a “lean” approach should be taken when deciding which attributes will be used. Don’t include a specific attribute unless you, your team, or your management has asked for that attribute and will be using that attribute in some manner to manage the project and set of needs and requirements. Wiegers (2006) makes the point that while attributes are all potentially useful, too many shouldn’t be created because “it takes effort to create and maintain them.” He says that attributes should “add value” and the systems engineering team must “be diligent about storing that information and keeping it current.” (Wiegers, 2006)

Attributes are used to manage projects more effectively. To use attributes to do so, they need to be accurate and current. As the old saying goes, “garbage in; garbage out.” Another applicable saying is “A time saving tool should not take more time to maintain than the time it saves.”

REFERENCE DOCUMENTS

The most current issues of applicable documents are listed in this section. Referenced documents are applicable only to the extent specified herein.

ANSI/AIAA G-043-2012e, *Guide to the Preparation of Operational Concept Documents*.

Carson, R. and Noel, R., “Formalizing Requirements Verification and Validation”, *INCOSE International Symposium IS2004*, July 2004.

Carson, R., E. Aslaksen, and R. Gonzales, “Requirements Completeness”, *INCOSE International Symposium IS2018*, July 2018.

Dick, J. and J. Chard, “The Systems Engineering Sandwich: Combining Requirements, Models and Design”, *INCOSE International Symposium IS2004*, July 2004.

Dick, J. and Llorens, J., "Using Statement-level Templates to Improve the Quality of Requirements", *International Conference on Software and Systems Engineering and Applications. ICSSEA 2012*, Paris, France.

Génova, G, Fuentes J.M., Llorens, J., Hurtado, O., and Moreno, V., “A Framework to Measure and Improve the Quality of Textual Requirements", *Requirements Engineering*, Vol 18, 2013.

Gilb, T. *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering and Software Engineering Management Using Planguage*, Elsevier Butterworth-Heinemann, 2005.

Heitmeyer, C., J. Kirby, and B. Labaw, “The SCR Method for Formally Specifying, Verifying, and Validating Requirements: Tool Support”, *Proceeding ICSE '97 Proceedings of the 19th international Conference on Software Engineering*, pp. 610-611, Boston, Massachusetts, USA — May 17 - 23, 1997.

Hull, E., K. Jackson, J. Dick, *Requirements Engineering*, Springer, 2011.

ISO/IEC, *ISO/IEC 29148 Systems and Software Engineering—Life Cycle Processes—Requirements Engineering, 2018*.

Ryan, M.J., “An Improved Taxonomy for Major Needs and Requirements Artefacts”, *INCOSE International Symposium IS2013*, June 2013.

Ryan, M., Wheatcraft, L, Dick, J, and Zinni, R, “An Improved Taxonomy for Definitions Associated with a Requirement Expression”, *Systems Engineering / Test and Evaluation Conference SETE2014*, Adelaide, 28-30 April 2014.

Ryan, M., Wheatcraft, L., “On the Use of the Terms Verification and Validation”, *INCOSE International Symposium IS2017*, July 2017.

Simplified Technical English (STE), Specification (ASD-STE100), ASD Belgium, http://www.asd-ste100.org/.

US Code of Federal Regulations: Title 21, Part 820, Quality System Regulation

Wheatcraft, L., Ryan, M., Dick, J. “On the Use of Attributes to Manage Requirements”, *Systems Engineering Journal*, Volume 19, Issue 5, September 2016, pp. 448–458.

Wiegers, K.E., *Software Requirements*, Redmond, WA: Microsoft Press, 2003.

Wiegers, K.E., *More About Software Requirements*, Redmond, WA: Microsoft Press, 2006.

A APPENDIX A. ACRONYMS AND ABBREVIATIONS

AFIS	Association Française d'Ingénierie Systèmes
API	Application Program Interface
BA	Business analyst
BNR	Business Needs and Requirements
BMRD	Business Management Requirement Document
BORD	Business Operations Requirement Document
CDR	Critical Design Review
CMMI	Capability Maturity Model - Integrated
ConOps	Concept of Operations
COTS	Commercial off-the-shelf
IAD	Interface Agreement Definition
ICD	Interface Control Document
IEC	International Electrotechnical Commission
IIDD	Internal Interface Definition Document
IRD	Interface Requirements Document
INCOSE	International Council on Systems Engineering
ISO	International Standards Organization
IV&V	Independent Verification and Validation
IV&V	Integration, Verification, and Validation
KDN	Key Driving Need
KDR	Key Driving Requirement
MBSE	Model Based Systems Engineering
OpsCon	Operational Concepts
PDR	Preliminary Design Review
PMI	Project Management Institute
RE	Requirements Engineer
RMT	Requirement Management Tool
RWG	Requirements Working Group
SE	Systems Engineering
STE	Simplified Technical English
SBP	Strategic Business Plan
SDK	Software Development Kit
SDR	System Design Review
SNR	Stakeholder Needs and Requirements
SOI	System of Interest
SOP	Standard Operating Procedure
SOW	Statement of Work
StRD	Stakeholder Requirement Document
StRS	Stakeholder Requirement Specification
SysM	Systems Modeling Language
SyRD	System Requirement Document
SyRS	System Requirement Specification
Vocab	Vocabulary

B APPENDIX B. REQUIREMENT PATTERNS

B.1 INTRODUCTION TO REQUIREMENT PATTERNS

The notion of requirement patterns (boilerplates or templates) was initially applied within the Future Surface Combatant (FSC) defence project in the United Kingdom in 1998 (Dick and Llorens, 2012) as an aid to solve several difficulties when writing different types of textual requirements (timeliness, etc.). Once requirement writers are shown a good example of how to form a requirement statement based on its type, the difficulty in writing a properly formed requirement was largely overcome.

The use of patterns also enables the development of natural language processing (NLP) tools or "digital assistants". These tools, along with a project ontology, can aid writers when writing need and requirement statements and helping ensure the rule and characteristics defined within this Guide are met.

"Structured and normalized" examples of types of requirements can be defined in patterns. A pattern may be structured as a sequential list of place-holders, including words, along with syntactic or semantic restrictions. These place holders are generally called pattern slots. The requirement text is written per a requirement pattern that is appropriate to what is to be communicated.

For example, the following example represents a requirement statement and the associated candidate pattern. In this example, the requirement pattern has 6 pattern slots.

The Power_Supply shall have an Availability greater than or equal to 98%.

<Subsystem> <shall> <have> <a: Optional> <PHYSICAL_PROPERTY> <OPERATOR> <PERCENTAGE>

Currently, the terminology for defining the abstractions that represent common natural language requirement structures at the syntactic and semantic level is not mature enough. Several approaches are used in the literature (see references at the end of this appendix) with very similar intent.

One common approach uses "requirement templates". The term "template" is most often used in the field of requirements management to refer to the structure and organization of a requirements document. The use of such templates helps ensure that the authors consider the complete range of concerns when organizing a set of requirement statements. For requirement statements, some authors have coined a more detailed term: "statement-level template" to avoid confusion with templates for requirement sets (Dick and Llorens, 2012).

In another approach Hull et al (2002) use "boilerplates" to refer to a grammatical structure for an individual textual requirement.

The term "pattern" has a clear meaning in software (Gamma et al, 1994) and recently systems engineering, usually representing a reusable structure that resolves a problem by instantiating and configuring this structure to become a solution. INCOSE and others promote this term for use in the system engineering life-cycle (Patterns Working Group at INCOSE). To align with these uses, this guide uses the term "patterns" to represent the concept of common natural language requirements structures representing syntactic and semantic information (restrictions and properties).

As this guide describes, many different factors contribute to the realization of high-quality requirement statements and sets of requirements. One important way to assist in achieving high quality of requirement statements and sets of requirements is by the proper definition and

agreement of a set of requirement patterns with which the requirements must comply. This appendix provides additional detail on requirement patterns.

The concept of a requirement patter' is mentioned in several sections of the guide—patterns are used to ensure that requirements and requirement sets obey the rules in this guide so that they have the desired characteristics.

B.2 BENEFITS OF USING PATTERNS TO FORM REQUIREMENT STATEMENTS

Ensuring that requirements conform to agreed-to patterns contributes to the requirement verification activity and helps to ensure the requirements properly communicate needs. In addition, writing requirements following a set of agreed-upon patterns also makes it easier to:

- write concise, easy to read and atomic requirement statements;
- find and classify requirements within a large document;
- find missing requirements (*completeness*);
- find inconsistences within a large set of requirements;
- find duplicated requirements (and finding and reusing requirements in general); and
- follow with other activities such as analysis and implementation.

In addition, following a set of agreed-upon patterns also leads to a consistent set of requirements, and the automatization of several requirement management activities such as:

- the semi-automatic requirement verification that requirement statements have the characteristics and conform to the rules defined in this guide;
- the verification of other consistency and completeness rules associated with systems engineering artifacts generated across all system life cycles (e.g. consistency of requirements versus design as stated in a models and consistent use of terms defined in the project ontology);
- the extraction of entities, properties, etc. from requirements or unstructured documents; and
- the translation of requirements to different languages.

Finally, requirement patterns can also facilitate the process of properly teaching a good and common way of writing requirements and leverage the use of consistent vocabulary.

B.3 BUILDING BLOCKS FOR REQUIREMENT PATTERNS

When it comes to defining the grammar of a requirement pattern, recursion has to be taken into account. In that sense, requirement patterns may be made up, if necessary, of smaller patterns (building blocks) that can also be further split in even smaller blocks.

This makes requirement patterns more modular and reusable, since different types of requirements can share some common building blocks, whilst also having unique elements.

For example, some special types of requirements may include (optionally) performance information. If the performance information is represented as a *second level* pattern, several advantages can be gathered:

- This second level pattern might be reused among different *top-level* patterns.
- Variations of the *second level* pattern can be defined yet extending the ‘expressivity’ of the entire catalog of patterns, with no need of changing the *top-level* ones.

Fortunately, the recursion referred in this appendix is not too deep; just a few levels should be enough to provide a set of flexible and reusable catalog of patterns. Usually, the complex patterns are formed by sub-patterns and the simple patterns are formed by simple restrictions, defining a set of building blocks. The way to construct these patterns catalog (bottom up, top down, iterative, etc.) is left to each organization based on their specific needs.

As an example, Figure B-1 shows the structure of sub-patterns used as building blocks that may be in a catalog of patterns.

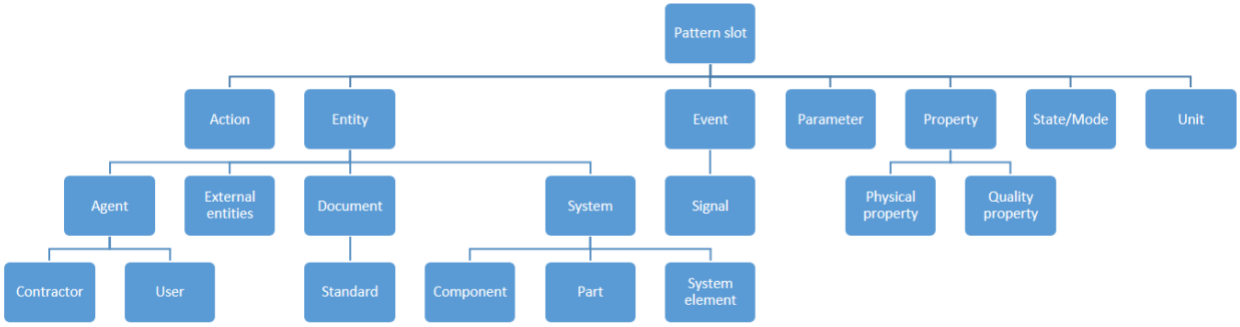


Figure B-1. Requirement pattern building blocks

Writing requirements based on agreed patterns is a good approach that will result in a consistent set of requirements. However, dealing with consistent patterns is largely pointless if the actual words and concepts used to specify a requirement are not used consistently. Every organization must solve this challenge within the scope of the knowledge management process. This common ontology must be implemented within the schema to be used to document and manage the requirement development and systems engineering process activities.

Other sources of requirement patterns are:

- Jeremy Dick, Juan Llorens, "Using Statement-level Templates to Improve the Quality of Requirements", *International Conference on Software and Systems Engineering and Applications. ICSSEA 2012*, Paris, France.
- Hull et al: *Requirements Engineering*, Springer, 2012.
- “EARS – Easy Approach to Requirements Syntax”, http://ieeexplore.ieee.org/document/5636542/
- The PABRE Catalog: http://www.upc.edu/gessi/PABRE/index.html
- ARTEMIS CRYSTAL EU Research Project: http://www.crystal-artemis.eu/

C APPENDIX C. COMMENT FORM

Reviewed Document:	
Name of submitter (first name & last name):	
Date Submitted:	
Contact information (email address):	
Type of submission (individual/group):	
Group name and number of contributors (if applicable)	***Please read examples carefully before providing your comments (and delete the examples provided.)***

Comment sequence number	Commenter name	Category (TH, TL, E, G)	Section number (eg, 2.1.1, no alpha)	Specific reference (e.g., paragraph, line, Figure no., Table no.)	Issue, comment and rationale *(rationale must make comment clearly evident and supportable)*	Proposed Changed/New Text -- *MANDATORY ENTRY* -- *(must be substantial to increase the odds of acceptance)*	Importance Rating (R = Required, I = Important, T = Think About for future version)

Submit comments to Working Group chair. Current WG chair will be listed at:

http://www.incose.org/techcomm.html

If this fails, comments may be sent to info@incose.org (the INCOSE main office), which can relay to the appropriate WG, if so requested in the comment cover page.

(INTENTIONALLY BLANK)

(BACK COVER)

INCOSE Publications Office
7670 Opportunity Road, Suite 220
San Diego, CA 92111-2222 USA