Guide for Writing Requirements

Document No.: INCOSE-TP-2010-006-01
Version/Revision: 1
Date: 17 April 2012

Prepared by:

Requirements Working Group
International Council on Systems Engineering (INCOSE)
7670 Opportunity Rd, Suite 220
San Diego, CA 92111-2222

NOTICE

This INCOSE Technical Product was prepared by the Requirements WG of the International Council on Systems Engineering (INCOSE). It is approved by the INCOSE Technical Operations Leadership for release as an INCOSE Technical Product.

Copyright (c) 2012 by INCOSE, subject to the following restrictions:

Author Use. Authors have full rights to use their contributions in a totally unfettered way with credit to the INCOSE Technical source. Abstraction is permitted with credit to the source.

INCOSE Use. Permission to reproduce and use this document or parts thereof by members of INCOSE and to prepare derivative works from this document for INCOSE use is granted, with attribution to INCOSE and the original author(s) where practical, provided this copyright notice is included with all reproductions and derivative works.

External Use. This document may not be shared or distributed to any non-INCOSE third party. Requests for permission to reproduce this document in whole or part, or to prepare derivative works of this document for external and commercial use should be addressed to the INCOSE Central Office, 2150 N. 107th St., Suite 205, Seattle, WA 98133-9009.

Electronic Version Use. Any electronic version of this document is to be used for personal, professional use only and is not to be placed on a non-INCOSE sponsored server for general use. Any additional use of these materials must have written approval from INCOSE Central.

Permissions. INCOSE has granted permission to member companies of the INCOSE Corporate Advisory Board to post and use this document internally, subject to the external use restriction.

Revision History

Revision	Revision Date	Change Description & Rationale
0	12/7/2009	Original
.1	1/1/2011	Revision to use standard INCOSE formatting
.2	30/1/2011	Revision to export from DOORS into standard INCOSE formatting
.3	30/1/2012	Revision to accommodate comments from review by INCOSE Technical Operations
.4	2/3/2012	Revision based on comments from INCOSE RWG leadership
1	17/4/2012	Revision based on comments from INCOSE RWG general membership

Preface

This document has been prepared and produced by a volunteer group of contributors within the International Council on Systems Engineering (INCOSE): the Requirements Working Group (RWG). 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. Subsequently, the document has been revised extensively following reviews by the RWG led by Mike Ryan and Jeremy Dick.

Authors

This document was prepared by the Requirements Working Group of the International Council on Systems Engineering. The authors who made a significant contribution to the generation of this Guide are:

Author	Organization
Jeremy Dick	Integrate Systems Engineering Ltd, UK
Michael Ryan	University of New South Wales, Canberra, Australia
Lou Wheatcraft	Requirements Experts, USA
Rick Zinni	Harris Corporation, USA
Kathy Baksa	Pratt & Whitney, USA
Jose L. Fernandez	Madrid Technical University
Gina R. Smith	Systems Engineering Global Inc
Christopher Unger	GE Healthcare

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 Rd., Suite 220
San Diego, CA 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

Table of Contents

1 INTRODUCTION

1.1 PURPOSE AND SCOPE

This guide is specifically about how to express textual requirements in the context of systems engineering. The aim is to draw together advice from a variety of existing standards into a single, comprehensive set of characteristics and rules.

The guide is not about the capture, elicitation or discovery of requirements, nor is it about requirements analysis and the ensuing models and designs. It focuses on how to express requirements clearly and precisely as text once they have been discovered, and in a form convenient for further analysis.

The guide does not mandate any particular boilerplate or template for a requirement statement, but focuses on the characteristics of, and rules for writing, such statements. Writers of requirements will need to conform to the existing template required by their organization.

Guidance on a wider range of requirements processes can be found in the INCOSE Requirements Working Group's REGAL database of good practice (www.incose.org/REGAL).

1.2 WHY TEXTUAL REQUIREMENTS?

Natural language is an imperfect way of expressing requirements. It is hard work to be clear, precise and to avoid ambiguity. However, it remains the only universal means of expression that covers the huge variety of concepts needed.

Alternatives to writing sentences to express requirements include:

- Diagrams as part of a modeling approach with well defined semantics, such as SysML.
- Tabular formats that provide template structures to collect and present requirements, such as Tom Gilb's *Planguage (*Gilb, T., 2005*).*

These other approaches are also imperfect: models do not yet cover the wide range of concepts needed, and tabular formats have presentational, traceability and management problems.

The reality is that textual requirements are still needed, if only to supplement other means of expression. The advantages of text are:

- There is no limitation on the concepts that can be expressed.
- Sentences and grammatical structure provide a means of tracing meaningful elements.

This guide refers solely to the expression of textual requirements.

1.3 AUDIENCE

The guide is intended for those whose role it is to *write* and to *review* textual requirements, and to some extent those who *read* requirements. Reading requirements requires less skill than writing, although it may help to understand why the statements are expressed in the way they are.

The guide is addressed to practitioners of all levels of experience. Someone new to this role should find specific guidance through the rules provided here, and those more experienced should be able to find new insights through the characteristics expressed, often absent from other standards.

Because this guide is an INCOSE product, access is restricted to members of INCOSE and to employees of organizations who are members of the INCOSE Corporate Advisory Board.

1.4 APPROACH

The guide presents underlying characteristics and practical rules. The characteristics express *why* the rules are a good idea, whereas the rules express *how* to write requirements. In other words, the characteristics lie in the problem domain and rules in the solution domain. Making this distinction is a principle of sound requirements management.

In this case, presenting 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-expressed requirements. The diverse nature of requirements means that the rules have to be adapted constantly 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 know the reason why we should.

1.5 CONCEPTS

The characteristics and rules are better understood if a number of concepts are first explained:

Levels of abstraction. Requirements exist at multiple levels of abstraction in systems engineering, ranging from the highest level expression of the customer need to the lowest level of design. How requirements are expressed differs through these layers, and therefore so do the rules for expressing them. As requirements are developed – and solutions designed – down through the layers of abstraction, we expect statements of requirement to become more and more specific. At the highest level, the ideal requirement is solution-free, and therefore not specific to a solution, and prescribes a range of possible solutions. At the lowest level, statements of requirement will

be entirely specific to the selected solution. In this guide we consider a single division between these layers: those requirements in the *problem* domain and those in the *solution* domain. While there is considerable iteration involved, the problem domain is considered to begin with a business need and end with the endorsement of the System Requirements Specification (SyRS) (ISO/IEC 29148), which is an implementation-independent set of requirements at the system level (expressing 'what' is required of the system). The definition of 'how' the system is to be implemented is then in the solution domain. While many of the rules contained herein apply equally to the problem and the solution domains, this guide is focused on the solution domain—that is, on the SyRS and the requirement sets below it.

Traceability of requirements. As design proceeds, requirements flow down through the levels of abstraction. There are two aspects of this flow-down: *allocation* and *decomposition*. Allocation of requirements occurs across level boundaries, i.e. they are allocated from level *n* to level *n*+1. For instance, a requirement on a sub-system will have been allocated from the system above. Bi-directional tracing should exist between an allocated requirement and its source. Once allocated, a requirement may be decomposed as part of the design process. This induces a many-to-many parent/child relationship between requirements, where the child requirements are in place to satisfy their parents. Bi-directional tracing should also exist between parent and child requirements. The terms *allocated*, *parent* and *child* will be used in this sense in this guide.

Types of requirement. Requirements that address capability and function may be expressed in a different manner to constraints and requirements specifying other system properties (often confusingly called "non-functional" requirements – a term that will not be used again in this guide). The guide is intended to cover the whole range of requirement types.

Requirements and specific disciplines. The way in which requirements are expressed is strongly influenced by the underlying discipline, such as verification, safety, security, reliability, etc, that owns the requirement. By ownership, we mean the discipline that either derived the requirement or that is responsible for satisfying it. The guide is intended to cover the expression of requirements from across disciplines, and across all stages of a product lifecycle, such as development, production, operation, maintenance, evolution and disposal.

1.6 STRUCTURE

This guide focuses on the writing of requirements and addresses the following aspects of requirements: the *characteristics of individual requirement statements*, the *characteristics of sets of requirements*, the *attributes of individual requirement statements*, and the *rules for individual requirement statements*.

Characteristics of requirement statements. In defining requirements, care should be exercised to ensure the requirement is appropriately crafted. The following characteristics of requirements statements (based on ISO/IEC 29148) are elaborated in this guide:

- *Necessary*.
- *Implementation Independent.*

- *Unambiguous.*
- *Complete.*
- *Singular.*
- *Feasible.*
- *Verifiable.*
- *Correct.*
- *Conforming.*

Characteristics of a set of requirement statements. In addition, the guide elaborates the following characteristics of a set of requirements, which should be addressed to ensure that the set of requirements collectively provides for a feasible solution that meets the stakeholder intentions and constraints. These include (based on ISO/IEC 29148):

- *Complete*.
- *Consistent*.
- *Feasible*.
- *Bounded*.

Attributes of requirement statements. In addition to the characteristics listed above, individual requirement statements may have a number of attributes attached to them (either as fields in a database, or through relationships with other artifacts):

- *Alignment with need*.
- *Alignment with design*.
- *Alignment with evidence*.
- *Priority*.

Rules for individual statements. The guide also provides a number of rules for writing individual statements, which support the characteristics and attribute of individual statements and the characteristics of a set of requirements.

2 CHARACTERISTICS OF REQUIREMENT STATEMENTS

2.1.1 C1 - *NECESSARY*

Every requirement statement is necessary.

Rationale:
If you can remove a requirement and still satisfy the problem, then that requirement is not needed. If the intent of the requirement is met by other requirements, then the requirement is not needed. If the author cannot communicate the reason for the requirement, the requirement is not needed. There is a cost associated with every requirement; unnecessary requirements can lead to non-value added work, additional cost and unnecessary risk.

Strategy:
There is no intrinsic characteristic of a requirement that indicates its necessity. It is about aligning every requirement to a genuine stakeholder expectation. At the highest level of requirements, this characteristic can only be assured by reviewing each requirement against the need, goals, and objectives as well as drivers, constraints, concepts and scenarios defined in the system scope. If the top level of requirements cannot be traced to the scope then it is not necessary. At lower levels, the necessity of a requirement can be expressed by tracing up to higher levels. The traceability of one layer of requirements to another can carry rationale covering the sufficiency and necessity of individual requirements. The inclusion of rationale for each requirement also aids in communicating the necessity and intent of the requirement.

2.1.2 C2 - *IMPLEMENTATION INDEPENDENT*

A requirement states what is required, not how the requirement is met.

Rationale:
Expressing a requirement in implementation terms has at least the following undesirable effects: - opportunities to consider other better implementations may missed; - the real problem to be solved may never be properly identified. - if the 'what' is not communicated at the current level, it cannot properly be allocated into the architecture of the next level.

Strategy:
Capture only those functions, characteristics or constraints that any solution must meet to be acceptable. A useful question to ask of a requirement is "for what purpose?" If the requirement is expressed in implementation terms, the answer to this question may be the real requirement. If the requirement is valid but at a lower level, determine what the appropriate level is and

document the requirement at that level.

Rules that help establish this characteristic:
R3 - /Precision/Subject Make the subject of the requirement appropriate to the layer in which the requirement lives.
R33 - /Abstraction/ProblemVocab If writing in the problem domain, express the problem to be solved in the vocabulary of the problem domain without recourse to solution.
R34 - /Abstraction/SolutionVocab If writing in the solution domain, express the system level solution in the vocabulary of the system.

2.1.3 C3 - *UNAMBIGUOUS*

A requirement statement lends itself to a single interpretation.

Rationale:
The intent of a requirement must be understood in the same way by the writer, the designer, and those doing verification and validation. Ambiguity leads to interpretations of a requirement not intended by the customer, and the ensuing problems, including project delay and even perhaps litigation and financial loss.

Strategy:
The possibility of ambiguity is reduced by applying a number of sound principles, including: - use correct grammar; - use defined terms and acronyms drawn from a central glossary; - use precise quantifiers and qualifiers; - use definite articles; - uniformly use agreed language forms; - avoid the use of ambiguous terms; - avoid the use of pronouns. Precision is achieved using a number of sound practices, including: - writing each requirement statement as a single sentence, expressed in the active voice, free of adjectives and adverbs and uncluttered by superfluous sub-clauses or auxiliary information, such as justifications, purposes and examples; - express conditions, performance or constraints in separate sub-clauses; - use appropriate non-textual representations such as figures or tables.

Rules that help establish this characteristic:
R1 - /Precision/UseDefiniteArticles Use definite articles.
R2 - /Precision/UseActiveVoice Use the active voice with the actor clearly identified.

R4 - /Precision/UseDefinedTerms Only use terms defined in the glossary.
R5 - /Precision/Quantify Avoid imprecise quantifiers.
R6 - /Precision/Units Use appropriate units when stating quantities.
R7 - /Precision/AvoidAdverbs Avoid adverbs.
R8 - /Precision/AvoidAdjectives Avoid adjectives.
R9 - /Precision/NoEscapeClauses Avoid escape clauses.
R10 - /Precision/NoOpenEnded Avoid open-ended clauses.
R11 - /Concision/NoInfinitives Avoid superfluous infinitives.
R12 - /Concision/SeparateClauses Use separate sub-clauses for each condition.
R13 - /NonAmbiguity/CorrectGrammar Use correct grammar.
R14 - /NonAmbiguity/CorrectSpelling Use correct spelling.
R15 - /NonAmbiguity/CorrectPunctuation Use correct punctuation.
R16 - /NonAmbiguity/Conjunction Use the construct 'X and Y' instead of 'both X and Y'.
R17 - /NonAmbiguity/AvoidAndOr Avoid the use of 'X and/or Y'.
R18 - /NonAmbiguity/Oblique Avoid the use of the oblique ("/") symbol.
R30 - /Conditions/ExplicitLists Express the propositional nature of a condition explicitly instead of giving lists of conditions.
R35 - /Quantifiers/Universals Use 'each' instead of 'all', 'any' or 'both' when universal quantification is intended.

2.1.4 C4 - *COMPLETE*

A requirement statement is complete in and of itself.

Rationale:
The effectiveness of several processes associated with requirements, such as decomposition, allocation, verification and validation, depends on a complete understanding of an individual statement without depending on other statements. Note, however, that a requirement can refer

to other documents, e.g., interface definition documents, standards, and regulations.
This characteristic also aids in the cognition process: each requirement can be understood in its own right without the overhead of having to understand a number of others.

Strategy:

While fully appreciating a requirement may require some context, the statement itself should be a complete sentence that does not require reference to other statements to be understood in its basic form.
Requirements should not refer to one another through use of pronouns.

Exceptions and relationships:

Completeness and *singularity* need to be balanced, and *modularity* has to be used to ensure that related requirements are grouped.

Rules that help establish this characteristic:

R6 - /Precision/Units Use appropriate units when stating quantities.
R7 - /Precision/AvoidAdverbs Avoid adverbs.
R8 - /Precision/AvoidAdjectives Avoid adjectives.
R9 - /Precision/NoEscapeClauses Avoid escape clauses.
R10 - /Precision/NoOpenEnded Avoid open-ended clauses.
R26 - /Completeness/AvoidPronouns Repeat nouns in full instead of using pronouns to refer to nouns in other requirement statements.
R27 - /Completeness/UseOfHeadings Avoid using headings to support explanation of subordinate requirements.
R29 - /Conditions/Explicit State applicability conditions explicitly instead of leaving applicability to be inferred from the context.

2.1.5 C5 - *SINGULAR*

A requirement statement addresses a single thought.

Rationale:

The effectiveness of several processes associated with requirements, such as decomposition, allocation, tracing, verification and validation depends on being able to identify singular statements. For instance, the verification and validation information defined against a requirement can be far more precise when that requirement addresses a single concern.

Strategy:
Keep the requirement limited to one function, characteristic or constraint. Understand how the statements fit into the traceability philosophy for the project. Use the project standard template for writing the requirements. Although a single requirement should consist of a single function, characteristic or constraint, it may have multiple conditions under which the requirement is to be met. Avoid the use of the word "and" when it ties together multiple thoughts (phrases in the sentence), each of which will be allocated and verified differently. The presence of the conjunction 'and' in a requirement statement should always prompt the writer to consider whether or not the statement is singular.

Exceptions and relationships:
Completeness and *singularity* need to be balanced, and *modularity* has to be used to ensure that related requirements are grouped.
Singularity and *Context* need to be balanced, because sometimes text is not the best way of expressing complex behavior, and referring to a model or design is the most concise and unambiguous way.

Rules that help establish this characteristic:
R19 - /Singularity/SingleSentence Write a simple affirmative declarative sentence, conditioned and qualified by relevant sub-clauses.
R20 - /Singularity/AvoidCombinators Avoid combinators.
R22 - /Singularity/AvoidPurpose Avoid phrases that indicate the purpose of the requirement.
R23 - /Singularity/AvoidParentheses Avoid parentheses containing subordinate text.
R24 - /Singularity/Enumeration Enumerate sets of entities explicitly instead of using generalizations.
R47 - /UniformLanguage/StyleGuide Use a project-wide style guide.

2.1.6 C6 - *FEASIBLE*

A requirement statement expresses something that is inherently feasible.

Rationale:
Inherently unachievable requirements, such as 100% reliability, are at best a waste of time, and at worst lead to needlessly expense solutions. Unrealistic requirements are usually important concerns that have not been properly quantified.

Strategy:

In general, it is not possible to tell whether an individual statement of requirement is feasible without considerable analysis of potential solutions. However, we can usually recognize and avoid requirements that are impossible or just plain unrealistic.
Feasibility/achievability can be measured in terms of cost, schedule, technology, and risk. Before allowing a requirement into your set, an assessment of achievability needs to be made using extant technologies and extant engineering methods. If not feasible within the stated measures, then why include the requirement in the set? Doing so can drive cost and schedule and can result in a requirement that will not be verifiable.

Rules that help establish this characteristic:

R28 - /Realism/AvoidAbsolutes
Avoid using unachievable absolutes.

2.1.7 C7 - *VERIFIABLE*

A requirement statement is verifiable.

Rationale:

Unless a requirement is in some way verifiable or testable, there is no way to tell if it has been satisfied. Different kinds of requirement are verifiable in different ways, and this will influence the way the requirement is written. Functional requirements, for instance, will typically require a test to verify that the correct behaviour exists in the product, so the behaviour needs to expressed clearly; whereas a performance requirement needs unambiguously to express the quantities associated with the performance.

Strategy:

The most usual causes for a requirement not to be verifiable are:
- no clear specification of correct functional behavior, conditions and states.
- lack of precision in the ranges of acceptable quantities.
- use of ambiguous terms.

Use a verification point-of-view to imagine yourself performing the verification event (analysis, inspection, demonstration or test).

Care should be taken to ensure that requirements are quantified 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 in a requirement absolute. A range of values is usually acceptable providing different performance levels.
2) Testing against a single absolute value is usually not possible, whereas testing against a defined range of values with upper and lower limits makes testing more tractable.

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 really intended.

Rules that help establish this characteristic:
R1 - /Precision/UseDefiniteArticles Use definite articles.
R2 - /Precision/UseActiveVoice Use the active voice with the actor clearly identified.
R4 - /Precision/UseDefinedTerms Only use terms defined in the glossary.
R5 - /Precision/Quantify Avoid imprecise quantifiers.
R6 - /Precision/Units Use appropriate units when stating quantities.
R7 - /Precision/AvoidAdverbs Avoid adverbs.
R8 - /Precision/AvoidAdjectives Avoid adjectives.
R9 - /Precision/NoEscapeClauses Avoid escape clauses.
R10 - /Precision/NoOpenEnded Avoid open-ended clauses.
R30 - /Conditions/ExplicitLists Express the propositional nature of a condition explicitly instead of giving lists of conditions.
R36 - /Tolerance/ValueRange Define the range of acceptable values associated with quantities.
R37 - /Quantification/Measurable Provide specific measurable performance targets.
R38 - /Quantification/TemporalIndefinite Define temporal dependencies explicitly instead of using indefinite temporal keywords.

2.1.8 C8 - *CORRECT*

A requirement statement is a correct expression of the stakeholder expectation.

Rationale:
This characteristic supplements the necessity of a requirement with the principle that the elements that it expresses - function and performance values - should be correct. This relates to the validation of the requirement against stakeholder expectations, to verification

that the designed and built system satisfies the requirement.

Strategy:
Ensure the statement reflects: - decomposition or derivation is based on a correct interpretation of the higher level requirement; - a correct understanding of the underlying goals (such as minimizing, maximizing with correct specified ranges); - the correct underlying analysis and assumptions. Use a defined requirement validation process to ensure correctness in the context of the individual requirement as well as the complete set of requirements.

Rules that help establish this characteristic:
R6 - /Precision/Units Use appropriate units when stating quantities.
R36 - /Tolerance/ValueRange Define the range of acceptable values associated with quantities.
R42 - /Traceability/ExternalReferences Make references to external documents specific to an identifiable element.

2.1.9 C9 - *CONFORMING*

A requirement statement conforms to standards selected as applicable to the organization.

Rationale:
When all requirements for a specific domain within the same organization have the same look and feel, each individual requirement is easier to write, understand, and review.

Strategy:
Apply organizational templates for requirement statements.

Rules that help establish this characteristic:
R43 - /UniformLanguage/Importance Use an agreed convention for distinguishing the relative importance of requirements.
R44 - /UniformLanguage/ConsistentTerms Use a glossary to define a consistent set of terms to be used in requirement statements.
R45 - /UniformLanguage/Acronyms Use an acronym list to define a consistent set of acronyms to be used in requirement statements.
R46 - /UniformLanguage/Abbreviations Use an abbreviation list to define a consistent set of abbreviations to be used in requirement statements.
R47 - /UniformLanguage/StyleGuide

Use a project-wide style guide.

3 CHARACTERISTICS OF SETS OF REQUIREMENTS

3.1.1 C10 - *COMPLETE*

A set of requirement statements represents a complete definition of the stakeholder expectations.

Rationale:

This characteristic is about ensuring that all features, functions and constraints are present across the whole set of requirements. Manifestly, it is a problem if there are omissions in the requirements set.
There are two questions to ask:
- Have all the requirements been captured?
- Have all the requirements been derived?

Strategy:

At the highest level, completeness can only fully be addressed through modeling and reviewing with the customer and all relevant stakeholders. When defining the scope of the project, ensure that all stakeholders are identified and involved; ensure scenarios are developed from all stakeholder viewpoints, for all lifecycle stages, and for both nominal and off-nominal operations. Identify all external interfaces and ensure each is defined and interface requirements are included for each interaction within a given interface.
For lower levels of abstraction, thorough and systematic tracing of lower-level statements to higher-level statements can provide confidence in achieving completeness, especially if the rationale for the relationships is also captured.
The use of a requirements heading structure can also aid in ensuring that all aspects are considered.
The goal is to clearly communicate stakeholder expectations via the minimum set of requirements that are necessary and sufficient and no more.

Rules that help establish this characteristic:

R40 - /Traceability/ParentChild
Establish traceability between each parent requirement and the set of child requirements that are derived from it.

3.1.2 C11 - *CONSISTENT*

A set of requirement statements represents a consistent expression of the stakeholder expectations.

Rationale:

Frequently, customer and other relevant stakeholder needs or design constraints are in conflict

with one another, and need to be reconciled. Conflicting requirements lead to an empty solution space, and if not identified early on in the development process, can lead to expensive rework. Additionally, even if individual requirements are unambiguous, the inconsistent use of terms and abbreviations in different requirements results in ambiguity in the requirement set.

Strategy:
Ensuring that identical or very similar requirements are not repeated in several places in a set of requirements is a particular challenge when the set of requirements is large. Similarly, it is a challenge spotting conflicts between requirements. It is hard to control this merely through the language used to express individual requirements, but it can be achieved by classifying and sorting requirements. One strategy is to classify requirements by the kinds of aspects they cover, e.g. safety, timeliness, functional area (bridges, level-crossings), etc. Then use sorting and filtering to bring otherwise diverse requirements together, and examine them for interactions, trade-offs and conflicts. Classification, therefore, is the only aspect that can be addressed through the expression of requirements rather than process. In general, it is only possible to ascertain whether such requirements are irreconcilable by engaging in modeling of potential solutions. However, it is possible to identify some expressions of requirements that are immediately irreconcilable. Requirements will often contain conflicting goals that will need to be traded against each other. Use a glossary to ensure consistent use of terms and abbreviations throughout the requirement set. Pay special attention to interface requirements to make sure they are consistent with the other systems your system of interest is interacting with.

Rules that help establish this characteristic:
R4 - /Precision/UseDefinedTerms Only use terms defined in the glossary.
R44 - /UniformLanguage/ConsistentTerms Use a glossary to define a consistent set of terms to be used in requirement statements.
R45 - /UniformLanguage/Acronyms Use an acronym list to define a consistent set of acronyms to be used in requirement statements.
R46 - /UniformLanguage/Abbreviations Use an abbreviation list to define a consistent set of abbreviations to be used in requirement statements.
R47 - /UniformLanguage/StyleGuide Use a project-wide style guide.

3.1.3 C12 - *FEASIBLE*

A set of requirements represents a feasible expression of the stakeholder expectations.

Rationale:
The complete set of requirements should be able to be satisfied by a solution that is obtainable/feasible within life cycle constraints (e.g., cost, schedule, technical, legal, regulatory). If infeasibility is not identified early on in the development process, it can lead to wasted effort and cost. While individual statements of requirement may seem feasible, they may not be so when placed in combination with others. That is, the combination of feasible individual requirements does not necessarily sum to a feasible set of requirements. 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 a 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 requirements are distributed throughout the requirement set, it is even more difficult for us to 'get our minds around' the feasibility of the set (see Characteristic C14—Structured).

Strategy:
In the solution space, there should be at least one and preferably multiple achievable solutions. The solution should be technically achievable and achievable within the constraints of the project (such as cost, schedule, technical and legal) and technology maturity level and advancement that aligns with the project goals.

Rules that help establish this characteristic:
R28 - /Realism/AvoidAbsolutes Avoid using unachievable absolutes.
R31 - /Uniqueness/Classify Classify the requirement according to the aspects of the problem or system it addresses.
R32 - /Uniqueness/ExpressOnce Express each requirement once and only once.
R36 - /Tolerance/ValueRange Define the range of acceptable values associated with quantities.
R37 - /Quantification/Measurable Provide specific measurable performance targets.

3.1.4 C13 - *BOUNDED*

A set of requirement statements is within a defined scope.

Rationale:

The set of requirements clearly communicate to the design team customer and other relevant stakeholder expectations that were defined in the system scope. All necessary requirements must be included; all irrelevant requirements must be excluded. The purpose of this characteristic is to avoid misunderstandings about the scope of the work, and to prevent resources being expended to achieve capabilities beyond those approved or agreed.

Strategy:
Develop and baseline scope before writing requirements. Scope clearly reflects stakeholder expectations. Once the scope is baselined (agreed to by the stakeholders) then the requirements can be written to reflect it. When the scope is formally captured, trace the requirements to the corresponding scope content. A context diagram is a useful part of defining scope, showing the interactions of the system under development with other systems, as well as communicating what is part of your system and what is not.

Rules that help establish this characteristic:
R44 - /UniformLanguage/ConsistentTerms Use a glossary to define a consistent set of terms to be used in requirement statements.
R45 - /UniformLanguage/Acronyms Use an acronym list to define a consistent set of acronyms to be used in requirement statements.
R46 - /UniformLanguage/Abbreviations Use an abbreviation list to define a consistent set of abbreviations to be used in requirement statements.
R47 - /UniformLanguage/StyleGuide Use a project-wide style guide.

3.1.5 C14 - *STRUCTURED*

A set of requirement statements is structured such that sub-sets of related requirement statements can be identified.

Rationale:
A well-structured document allows an understanding of the whole set of requirements to be assimilated without undue cognitive loading on the reader. Despite what has been stated about the completeness of individual statements of requirement, it is often easier to understand a requirement 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 requirements is a vital tool in identifying repetition and conflict between requirement statements.

Strategy:
Use the hierarchical decomposition of functions as a cognitive grouping for requirements. However, this decomposition should be limited to at most seven subcategories per branch and

the subcategories should be chosen as natural (and not arbitrarily chosen) entities. Organizations should develop a template document for their domain that defines the structure expected for systems developed for that domain. Use a requirements management tool configured to permit the identification of sets of related requirements.

Exceptions and relationships:
Completeness and *singularity* need to be balanced, and *modularity* has to be used to ensure that related requirements are grouped.

Rules that help establish this characteristic:
R47 - /UniformLanguage/StyleGuide Use a project-wide style guide.
R48 - /Modularity/Dependents Group mutually dependent requirements together.
R49 - /Modularity/RelatedRequirements Group related requirements together.

4 ATTRIBUTES OF REQUIREMENTS STATEMENTS

4.1.1 A1 - *ALIGNMENT WITH NEED*

The satisfaction relationship between levels of requirements is documented.

Rationale:
The purpose of this characteristic is manifold: - to support an understanding of exactly how a solution solves the problem; - to avoid an over-engineered solution containing features that add no value; - to avoid an under-engineered solution that fails to address parts of the problem; - to inform the process of change management across a complex set of requirements defined through multiple layers of abstraction.

Strategy:
The strategy here is to trace individual statements of requirement to the higher-level statements that they serve to satisfy. Capturing the rationale for the satisfaction relationship is also a valuable discipline. Determine that sets of lower-level requirements are necessary and sufficient when implemented to satisfy the higher-level requirements they trace to.

Exceptions and relationships:
Singularity and *Context* need to be balanced, because sometimes text is not the best way of expressing complex behavior, and referring to a model or design is the most concise and unambiguous way.

Rules that help establish this characteristic:

R32 - /Uniqueness/ExpressOnce Express each requirement once and only once.
R39 - /Traceability/UniqueReference Provide a unique reference for each requirement.
R40 - /Traceability/ParentChild Establish traceability between each parent requirement and the set of child requirements that are derived from it.
R42 - /Traceability/ExternalReferences Make references to external documents specific to an identifiable element.

4.1.2 A2 - *ALIGNMENT WITH DESIGN*

The relationship between the requirements and the design is documented.

Rationale:
The way in which requirements flow down through the layers of abstraction should reflect the design, and therefore the flow-down structure should be connected to the design. The documentation of such relationships also aids analysis of the impact on design of changing requirements. This relationship will also help during integration and verification where proof that the system as designed and built meets the requirements.

Strategy:
The strategy here is to trace design artifacts to those individual requirements (or sets of requirements) that influence the development of those artifacts. It also arises that design artifacts (such as architectures) influence the allocation and derivation of requirements, and in these cases the requirements should be traced to those artifacts.

Rules that help establish this characteristic:
R25 - /Singularity/Context When a requirement is related to complex behavior, refer to the supporting design or model.
R32 - /Uniqueness/ExpressOnce Express each requirement once and only once.
R40 - /Traceability/ParentChild Establish traceability between each parent requirement and the set of child requirements that are derived from it.
R41 - /Traceability/Verification Every requirement statement should be linked to the verification artifacts that contribute to its verification.
R42 - /Traceability/ExternalReferences Make references to external documents specific to an identifiable element.

4.1.3 A3 - *ALIGNMENT WITH EVIDENCE*

The relationship between requirements and verification artifacts is documented.

Rationale:
The requirements set not only supports the design, but also drives the collection of evidence needed to support assurance, conformance and acceptance. Evidence is collected through the planning and execution of validation and verification methods, such as analysis, inspection, demonstration and test. The relationship between requirements and validation/verification artifacts should therefore be documented, so that the adequacy, necessity and outcomes of the verification and validation activities can be accessed. The documentation of such relationships also aids analysis of the impact on verification and validation of changing requirements.

Strategy:
The strategy here is to trace individual requirements to those validation and verification artifacts that provide evidence for the satisfaction of that requirement and, in turn, to the results of the validation and verification actions.

Rules that help establish this characteristic:
R32 - /Uniqueness/ExpressOnce Express each requirement once and only once.
R41 - /Traceability/Verification Every requirement statement should be linked to the verification artifacts that contribute to its verification.
R43 - /UniformLanguage/Importance Use an agreed convention for distinguishing the relative importance of requirements.
R47 - /UniformLanguage/StyleGuide Use a project-wide style guide.

4.1.4 A4 - *PRIORITY*

A requirement statement possesses an indication of relative priority.

Rationale:
The goal of satisfying all requirements may not be achievable within a given timeframe with the resources available. In addition, when a project is faced with budget overruns, schedule slips, or unanticipated technical challenges, the project may be forced to de-scope. An indication of priority helps in deciding those requirements that will best be used to drive the system design (e.g., assigning weights to criteria in a trade study). Prioritized requirements also provide trade space when design decisions are being made.

Strategy:
When a requirement is written, ensure that the priority is identified. Prioritize requirements so that those most critical to (early) project success can be addressed first and those with the least impact to project success can wait for later interactions, or be accepted with a lower standard of compliance. Importance and urgency are other attributes that serve this purpose.

Priority of requirements flows down with the requirements.

Rules that help establish this characteristic:
R43 - /UniformLanguage/Importance Use an agreed convention for distinguishing the relative importance of requirements.

5 RULES FOR INDIVIDUAL STATEMENTS

5.1 PRECISION

5.1.1 R1 - /PRECISION/USEDEFINITEARTICLES

Use definite articles.

Elaboration:
The definite article is 'the'; the indefinite article is 'a'. Use of the indefinite article can lead to ambiguity. For example, if the 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--for example, babies are arguably users of baby food, but the system would fail if the test agency sought to verify that a baby could order, receive, open and serve (and even independently consume) baby food. On the other hand, if the requirement refers to 'the User', the reference is explicitly to the nature of user defined in the glossary.

Examples:
Unacceptable: The system shall provide a time display. *{This is not acceptable because it is ambiguous--will any time 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 '10:00am' (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.}* *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 of the more-general term 'current time'.}*

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

5.1.2 R2 - /PRECISION/USEACTIVEVOICE

Use the active voice with the actor clearly identified.

Elaboration:
The active voice requires that the actor/entity performing the action is the subject of sentence, since the onus for satisfying the requirement is on the subject, not the object of the sentence. If the actor/entity responsible for the system is not identified explicitly, it is unclear who/what should perform the action. Further, verification of the requirement is very difficult; largely because it is the responsible actor/entity that is to be the subject of the verification activity. The subject also helps ensure the requirement is stated at the appropriate level consistent with the actor name (see R3 /Precision/Subject).

Examples:
Unacceptable: The Identity of the Customer shall be confirmed. *{This is not acceptable because it does not identify who/what is responsible/accountable for confirming the identity.}* *Acceptable*: The Accounting_System shall confirm the Identity of the Customer. *{Note that 'Accounting_System', 'Identity', 'confirm' and 'Customer' must be defined in the glossary since there are a number of possible interpretations of these terms.}*

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

5.1.3 R3 - /PRECISION/SUBJECT

Make the subject of the requirement appropriate to the layer in which the requirement lives.

Elaboration:
The subject of a requirement statement is an indicator of its level. System requirements with "The system shall ..."; subsystem requirements with "The subsystem shall ...", etc. Ask yourself if the subject of a requirement is the right one, or would the requirement be better expressed at a lower or higher level. Also ask if this is the appropriate level to verify that the requirement has been met.

Examples:
System requirements with "The <system> shall ...". *Subsystem requirements with* "The <subsystem> shall ...".

Characteristics that are established by this rule:
C2 - Implementation Independent

5.1.4 R4 - /PRECISION/USEDEFINEDTERMS

Only use terms defined in the glossary.

Elaboration:
Most languages are extremely rich with almost every word having a number of synonyms, each with a subtly different meaning. In requirements, shades of meaning will most likely lead to ambiguity and to difficulty in verification. The use of a glossary allows the reader of a requirement to know exactly what the writer meant when a particular word was chosen. A standard should be agreed to make the use of glossary terms identifiable in the requirements text; 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.

Examples:
Unacceptable: The arrivals board shall display continuously the current time. *{This is not*

acceptable because it is ambiguous--what is 'current', in which time zone, to what degree of accuracy, in what format?}
Acceptable: The Arrivals_Board shall display the Current_Time. *{Note that 'Arrivals_Board' and 'Current _Time' must then be defined in the glossary, the latter in terms of accuracy, format, and time zone.}*

Characteristics that are established by this rule:
C3 - Unambiguous
C7 - Verifiable
C11 - Consistent

5.1.5 R5 - /PRECISION/QUANTIFY

Avoid imprecise quantifiers.

Elaboration:
Requirements should be quantified precisely. Avoid words that provide vague quantification, such as 'some', 'any', 'several', 'many', 'a lot of', 'a few', 'approximately', 'almost always', 'very nearly', 'nearly', 'about', 'close to', 'almost', 'approximate', 'significant', 'flexible', 'expandable', 'typical', 'sufficient', 'adequate', 'appropriate', 'efficient', 'effective', 'proficient', 'reasonable'. Care should also be taken to avoid unspecified units and unspecified value ranges.

Examples:
Unacceptable: The Flight_Information_System shall display the current altitude to approximately 1 meter resolution. *{This is not acceptable because it is imprecise. What is 'approximate' in the context of a distance of 1m? Who has the option of deciding what is 'approximate'. How will 'approximately' be verified?}* *Acceptable*: The Flight_Information_System shall display Current_Altitude plus or minus 1 meter. *{Note that 'Current_Altitude' must be defined in the glossary since there are a number of possible interpretations of the term.}*

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

5.1.6 R6 - /PRECISION/UNITS

Use appropriate units when stating quantities.

Elaboration:
All numbers should have some kind of units explicitly stated.

Examples:
Unacceptable: The Circuit_Board shall have a storage temperature or not more than 30 degrees. *Acceptable*: The Circuit_Board shall have a storage temperature or not more than 30 degrees Celsius.

5.1.7 R7 - /PRECISION/AVOIDADVERBS

Avoid adverbs.

Elaboration:
Adverbs qualify actions in some way. Avoid vague adverbs, such as 'usually', 'approximately', 'sufficiently', 'typically'. Vague adverbs can lead to ambiguous, unverifiable requirements that do not reflect accurately the stakeholder expectations.

Examples:
Unacceptable: The Flight_Information_System shall usually be on line. *Acceptable*: The Flight_Information_System shall have an Availability of at least xx% over a period of at least yy hours. *{Note that 'Availability' must be defined in the glossary since there are a number of possible ways of calculating that measure.}*

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

5.1.8 R8 - /PRECISION/AVOIDADJECTIVES

Avoid adjectives.

Elaboration:
Adjectives qualify entities in some way. Avoid vague adjectives such as 'ancillary', 'relevant', 'routine', 'common', 'generic' and 'customary'. Vague adjectives can lead to ambiguous, unverifiable requirements that do not reflect accurately the stakeholder expectations.

Examples:
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.}*

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

5.1.9 R9 - /PRECISION/NOESCAPECLAUSES

Avoid escape clauses.

Elaboration:
An escape clause gives an excuse to the supplier not to bother with a requirement. They usually provide vague conditions or possibilities, using phrases such as 'so far as is possible', 'as little as possible', 'as much as possible', 'if it should prove necessary', 'where possible' and 'if practicable'. Escape clauses can lead to ambiguous, unverifiable requirements that do not reflect accurately the stakeholder expectations.

Examples:
Unacceptable: The GPS shall, where there is sufficient space, display the location of the User_Location. *Acceptable*: The GPS shall display the User_Location. *{Note that 'GPS' and 'User_Location' must be defined in the glossary.}*

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

5.1.10 R10 - /PRECISION/NOOPENENDED

Avoid open-ended clauses.

Elaboration:
Open-ended clauses say that there is more required without stating exactly what. Avoid phrases such as 'including but not limited to', 'etc.' and 'and so on'. Open-ended clauses can lead to ambiguous, unverifiable requirements that do not reflect accurately the stakeholder expectations. Use of open-ended clauses also violates the one thought rule. If more cases are required, then include additional requirements that explicitly state those cases.

Examples:
Unacceptable: The ATM shall display the Customer Account_Number, Account_Balance, and so on. *Acceptable*: *(Split into as many requirements as necessary to be complete):* 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. The ATM shall display the Customer .....

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

C7 - Verifiable

5.2 CONCISION

5.2.1 R11 - /CONCISION/NOINFINITIVES

Avoid superfluous infinitives.

Elaboration:

We sometimes see a requirement that contains more verbs than are 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 has the system met the requirement? No.

Examples:

Unacceptable: The Weapon_Subsystem shall be able to store the location of all Ordnance.
Acceptable: The Weapon_Subsystem shall store the location of all Ordnance. *{Note that the terms 'Weapon_Subsystem' and 'Ordnance' must be defined in the glossary.}*

Characteristics that are established by this rule:

C3 - Unambiguous

5.2.2 R12 - /CONCISION/SEPARATECLAUSES

Use separate sub-clauses for each condition.

Elaboration:

Each requirement should have a main verb describing a basic function or need. The main sentence may then be supplemented by sub-clauses that provide conditions, performance values or constraints. A single, clearly identifiable sub-clause should be used for each condition expressed.

Examples:

This interleaves clauses in such a way that the object of the sentence is separated from the verb:
The Navigation_Beacon shall provide Augmentation_Data for 8-20 meters accuracy at least 99.7% of the time to each Maritime_User during Harbor/Harbor Approach (HHA) maneuvering.
This places the basic function together in an unbroken clause followed by the sub-clause describing performance:
The Navigation_Beacon shall provide Augmentation_Data to each Maritime_User engaged in Harbor/Harbor_Approach_maneuvering (HHA), at an accuracy of 8-20 meters at least 99.7% of the time.
{Note that:
- 'Navigation_Beacon', 'Maritime_User' and 'Harbor_Harbor_Approach_manoeuvring (HHA)' must be defined in the glossary.
- the two quantities (accuracy and availability) are not independently verifiable, and so remain

together in a single requirement.}

Characteristics that are established by this rule:

C3 - Unambiguous

5.3 NON-AMBIGUITY

5.3.1 R13 - /NONAMBIGUITY/CORRECTGRAMMAR

Use correct grammar.

Elaboration:

Incorrect grammar clouds understanding.
Particular care must be taken when translating requirements from one language to another, when sentence structure differs depending on the language in which the original requirement was written.

Examples:

Unacceptable: The Weapon_Subsystem shall storing the location of all ordnance. *{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.}*

Characteristics that are established by this rule:

C3 - Unambiguous

5.3.2 R14 - /NONAMBIGUITY/CORRECTSPELLING

Use correct spelling.

Elaboration:

Incorrect spelling can lead to confusion.

Examples:

Unacceptable: The Weapon_Subsystem shall store the location of all ordinance. *{It is unlikely that the Weapon_Subsystem is interested in regulations.}*
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.}*

Characteristics that are established by this rule:

C3 - Unambiguous

5.3.3 R15 - /NONAMBIGUITY/CORRECTPUNCTUATION

Use correct punctuation.

Elaboration:
Incorrect punctuation can cause confusion between sub-clauses in a requirement.

Examples:
Unacceptable: The Navigation_Beacon shall provide Augmentation_Data to each Maritime_User, engaged in Harbor/Harbor_Approach_maneuvering (HHA) at an accuracy of 8-20 meters at least 99.7% of the time. *{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 8-20 meters, at least 99.7% of the time. *{The positioning of the commas now makes it clear that the accuracy and availability relate to the data.}*

Characteristics that are established by this rule:
C3 - Unambiguous

5.3.4 R16 - /NONAMBIGUITY/CONJUNCTION

Use the construct 'X and Y' instead of 'both X and Y'.

Elaboration:
The expression 'X and Y' should be understood to mean both of them without having to add the word 'both'. Adding the word 'both' may lead the reader to think that something different is meant.

Examples:
Unacceptable: The Engine_Management_System shall disengage the Speed_Control_Subsystem when both the Cruise_Control is engaged and the Driver is applying the Accelerator. *Acceptable*: The Engine_Management_System shall disengage the Speed_Control_Subsystem when [the Cruise_Control is engaged AND the Driver is applying the Accelerator].

Characteristics that are established by this rule:
C3 - Unambiguous

5.3.5 R17 - /NONAMBIGUITY/AVOIDANDOR

Avoid the use of 'X and/or Y'.

Elaboration:
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.

Examples:
Unacceptable: When the Clutch is disengaged and/or the Driver is applying the Brake, the Engine_Management_System shall disengage the Speed_Control_Subsystem.

Acceptable as two requirements:
- When the Clutch is disengaged, the Engine_Management_System shall disengage the Speed_Control_Subsystem.
- When Driver is applying the Brake, the Engine_Management_System shall disengage the Speed_Control_Subsystem.

Characteristics that are established by this rule:

C3 - Unambiguous

5.3.6 R18 - /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 in requirements. The slash symbol (like the construct 'and/or') can lead to ambiguous requirements that do not reflect accurately the true customer needs.

Examples:

Unacceptable: The GPS shall retain 100% of its functions while being exposed to vibration/shock.
Acceptable (Split into two requirements):
The GPS shall retain 100% of its functions while being exposed to vibration of intensity xx (units) for a duration of zz (units).
The GPS shall retain 100% of its functions after being exposed to a single shock of intensity yy (units) of duration zz (units).

Characteristics that are established by this rule:

C3 - Unambiguous

5.4 SINGULARITY

5.4.1 R19 - /SINGULARITY/SINGLESENTENCE

Write a simple affirmative declarative sentence, conditioned and qualified by relevant sub-clauses.

Elaboration:

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.

ISO/IEC 29148 states that a typical sentence form for a functional requirements is 'When <condition clause>, the <subject clause> shall <action verb clause> <object clause> <optional qualifying clause>.

The use of this style, however, means that all requirements do not start with the standard construct “The <subject clause> shall ....”. To retain that structure, therefore, there are two other

styles in common use:

1) Some styles retain the standard construct “The <subject clause> shall ....” and place the condition immediately after the verb clause (‘The <subject clause> shall, when <condition clause>, <action verb clause> <object clause> <optional qualifying clause>.’) because the condition is a qualification of the verb and should therefore occur immediately after it. Some style guides eschew this format, however, preferring to start all requirements without interruption of the sequence of subject clause, verb clause and object clause.

2) Another approach is to retain the construct “The <subject clause> shall ....”, and place the condition at the end of the sentence (‘The <subject clause> shall <action verb clause> <object clause> <optional qualifying clause>, when <condition clause>.’). While this retains the intimacy between the subject, verb and object clauses, some styles prefer not to have such a large separation between the verb clause and the condition that relates to it.

Although the ISO/IEC 29148 style is preferred for consistency internationally, any one of the three styles is acceptable so long as the same construct is used consistently throughout the requirement set.

A typical form for a constraint is ' When <condition clause>, the <subject clause> shall not <verb clause> <optional qualifying clause>.'

Examples:

Acceptable:
When the temperature of water in the Boiler is less than 85 °C, the Control_Subsystem shall open the Inlet_Valve in less than 3 seconds.
The Control_Subsystem shall, when the temperature of water in the Boiler is less than 85 °C, open the Inlet_Valve in less than 3 seconds.
The Control_Subsystem shall open the Inlet_Valve in less than 3 seconds, when the temperature of water in the Boiler is less than 85 °C.

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 requirements. Additionally, it contains two occurrences of the pronoun 'it' ambiguously referring to different things (see Rule 24), the terms 'exceeds' and 'has reduced' are ambiguous, and the action verb must be 'shall', not 'will'.}*
Acceptable (Split into two requirements and quantify performance) : The Control_Subsystem shall close the Inlet_Valve in less than 3 seconds when the temperature of water in the Boiler is greater than 85 °C.
The Control_Subsystem shall open the Inlet_Valve in less than 3 seconds when the temperature of water in the Boiler is less than 85 °C.
{Note use of the glossary to define terms.}

Exceptions and relationships:
Every requirement should have a main clause with a main verb. However, additional sub-clauses with auxiliary verbs 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*'. Note that, if performance attributes need to be verified separately, then they should be expressed as sub-clauses in separate requirements.

Characteristics that are established by this rule:
C5 - Singular

5.4.2 R20 - /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.

Examples:
Unacceptable: The user shall either be trusted or not trusted. *{This real example is unacceptable for a number of reasons. The intention is that user should be classified in one of two ways.}* *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 function 'command the sidelights' knows the demands of the lights, front and back fog lights, to ensure the regulatory consistency of lights ignition: the ignition of sidelights is maintained during the ignition of the side lamps, or the lights, or the fog lights or the back fog lights or a combination of these lights." *{Again, this real example is unacceptable for all sorts of reasons. 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 holds: the Side_Lamps are illuminated, the Head_Lamps are illuminated, the Front_Fog_Lamps are illuminated, the Rear_Fog_Lamps are illuminated." *{Rule R24 has also been applied to remove combinators and clarify the exact conditions.}*

5.4.3 R21 - /Singularity/AvoidCombinators/ExceptInConditions

Use an agreed typographical device to indicate the use of propositional combinators for expressing a logical condition within a requirement.

Elaboration:
Sometimes the use of 'and' and 'or' is unavoidable. This is because there are times when several things need to be present to achieve a correct or desired result. This is often the case when compound conditions have to be defined. If such combinators have to be used, then a convention for this use should be agreed and stated explicitly at the start of the requirement document. *Examples of conventions:* 1. Place conjunctions in italics 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.

Examples:
Unacceptable: The Controller shall close values Valv1 and Valv2, or Valv3 when the temperature of water in the Boiler exceeds 95 ºC. *Acceptable*: The Controller shall, when the temperature of water in the Boiler is greater than 95 ºC, close the following valves: [EITHER [Valv1 AND Valv2] OR Valv3]. *{In this example, the 'when' sub-clause has been positioned after the verb, allowing the valve expression to be positioned at the end. Note use of the glossary to define terms.}*

Exceptions and relationships:
While the rule **R20: /Singularity/NoCombinators** states that 'and' and 'or' are to be avoided, such words may be used to express a logical condition within a single requirement, as described in **R21: /Singularity/NoCombinators/ExceptInConditions.**
While the rule **R23: /Singularity/NoParentheses** states that brackets are to be avoided, the rule **R21: /Singularity/NoCombinators/ExceptInConditions** suggests that brackets may be used as part of a convention for disambiguating conditions**.**

5.4.4 R22 - /Singularity/AvoidPurpose

Avoid phrases that indicate the purpose of the requirement.

Elaboration:
The text of a requirement does not have to carry around extra baggage such as the purpose of its existence. Expressions of purpose are often indicated by the presence of phrases such as 'in order to', 'so that', 'thus allowing'. This extra information should be included in the requirement rationale statement.

Examples:
Unacceptable: The Database shall store Account_Balance information until it is at least 10 years

old in order to satisfy the demands of the auditors.
Acceptable: The Audit_System shall store Account_Balance information relating to at least the last 10 years.

Characteristics that are established by this rule:
C5 - Singular

5.4.5 R23 - /Singularity/AvoidParentheses

Avoid parentheses containing subordinate text.

Elaboration:
If the text of a requirement contains brackets, then it usually indicates the presence of superfluous information that can simply be removed or communicated in the rationale. Other times, brackets introduce ambiguity.
Conventions for the use of brackets may be agreed for specific purposes, but such conventions should be documented at the start of the 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.)
Acceptable: The Control_Unit shall disconnect power from the Boiler when the water temperature exceeds 85 °C.

Exceptions and relationships:
While the rule **R23: /Singularity/NoParentheses** states that brackets are to be avoided, the rule **R21: /Singularity/NoCombinators/ExceptInConditions** suggests that brackets may be used as part of a convention for disambiguating conditions**.**

Characteristics that are established by this rule:
C5 - Singular

5.4.6 R24 - /Singularity/Enumeration

Enumerate sets of entities explicitly instead of using generalizations.

Elaboration:
If lists of items are given, a requirement should be written for each item. Generalizations are ambiguous.

Examples:
Unacceptable: The software packages (i.e., the HLT algorithm, the selection control, the data access) shall be documented for the maintainer.
Acceptable (3 requirements):
The maintenance documentation shall include descriptions of each HLT algorithm.
The maintenance documentation shall include descriptions of the selection control.

The maintenance documentation shall include descriptions of the data access.

Characteristics that are established by this rule:

C5 - Singular

5.4.7 R25 - /SINGULARITY/CONTEXT

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

Elaboration:

Sometimes you can tie yourself in knots trying to express a complicated requirement in words, and it is better simply to refer to a model or diagram.

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.
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.}*

Characteristics that are established by this rule:

A2 - Alignment with design

5.5 COMPLETENESS

5.5.1 R26 - /COMPLETENESS/AVOIDPRONOUNS

Repeat nouns in full instead of using pronouns to refer to nouns in other requirement statements.

Elaboration:

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 words; but when writing requirements, pronouns are effectively cross-references to nouns in other requirements and, as such, should be avoided. Repeat the proper nouns where necessary.

Examples:

Unacceptable: The controller shall send the driver his itinerary for the day.
Acceptable: The Controller shall send the Driver_Itinerary for the day to the Driver. *{Note use of the glossary to define terms and to be explicit about the relationship between the driver and the itinerary for that particular driver.}*

Characteristics that are established by this rule:

C4 - Complete

5.5.2 R27 - /COMPLETENESS/USEOFHEADINGS

Avoid using headings to support explanation of subordinate requirements.

Elaboration:
It is a common mistake to use the heading under which the requirement falls to contribute to the explanation of the requirement.The requirement should be complete in and of itself and not require the neading for it to make sense.

Examples:
Example heading: Alert Buzzer Requirements *Unacceptable:* It shall sound for no longer than 20 minutes. *Acceptable:* The Alert Buzzer shall sound for no longer than 20 minutes.

Characteristics that are established by this rule:
C4 - Complete

5.6 REALISM

5.6.1 R28 - /REALISM/AVOIDABSOLUTES

Avoid using unachievable absolutes.

Elaboration:
An absolute, such as '100% reliability', is hardly ever achievable. Think ahead to verification, how would you prove 100% reliability? Even if you could build such a system, could you afford to do so?

Examples:
Unacceptable: The system shall have 100% availability. *Acceptable*: The system shall have an Availability of greater than or equal to 98%. *{Note use of the glossary to define terms.}*

Characteristics that are established by this rule:
C6 - Feasible
C12 - Feasible

5.7 CONDITIONS

5.7.1 R29 - /CONDITIONS/EXPLICIT

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

Elaboration:
Sometimes requirements are only applicable under certain conditions. If so, the condition should be repeated in the text of each requirement, rather than stating it separately in a paragraph of

the document.

Examples:

Unacceptable: In the event of a fire:
Power to all electromagnetic magnetic fire door catches shall be turned off.
All security entrances shall be set to Free_Access_Mode.
All fire escape doors shall be unlocked.

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

Characteristics that are established by this rule:

C4 - Complete

5.7.2 R30 - /CONDITIONS/EXPLICITLISTS

Express the propositional nature of a condition explicitly instead of giving lists of conditions.

Elaboration:

When a list of conditions is given in a requirement, it may not be clear whether all the conditions must hold (a conjunction) or any one of them (a disjunction). The wording of the requirement should make this clear.

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;
- the Audit _Clerk is the reviewer.

Acceptable if interpreted as a disjunction: The Audit_System shall allow the Audit_Clerk to change the status of the action item when any one 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 all of the following conditions hold:
- the Audit _Clerk originated the item;
- the Audit _Clerk is the actionee;
- the Audit _Clerk is the reviewer.

Characteristics that are established by this rule:

C3 - Unambiguous

C7 - Verifiable

5.8 UNIQUENESS

5.8.1 R31 - /UNIQUENESS/CLASSIFY

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

Elaboration:
By classifying requirements in various ways, it is possible to regroup requirements to help identify potential duplications and conflicts. The ability to view groups of requirements can also assist in identifying what requirements may be missing.

Examples:
Classification can be undertaken in a number of ways. In a requirements management system, a field in the database can be used to classify each requirement as one or more types; a report can then be obtained for all requirements of a certain type, allowing the identification of duplication, conflict, or absence of requirements.

Characteristics that are established by this rule:
C12 - Feasible

5.8.2 R32 - /UNIQUENESS/EXPRESSONCE

Express each requirement once and only once.

Elaboration:
Care should be taken to avoid including the same requirement more than once, either as a duplicate or in similar form. Exact duplicates are relatively straightforward to identify; finding similar requirements 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. For example: "The system shall provide report of financial transactions" and "The system shall provide a financial report" are overlapping in that the first is a subset of the second. Avoidance of repetition can be assisted by classification so that a subset of requirements can be compared.

Characteristics that are established by this rule:
C12 - Feasible
A1 - Alignment with need
A2 - Alignment with design
A3 - Alignment with evidence

5.9 ABSTRACTION

5.9.1 R33 - /ABSTRACTION/PROBLEMVOCAB

If writing in the problem domain, express the problem to be solved in the vocabulary of the problem domain without recourse to solution.

Elaboration:
Every system endeavor should have a level of requirements that capture the problem to be solved without involving solutions. Of each requirement statement, ask yourself "for what purpose?" (Notice that this question is subtly different from simply asking "Why?", and encourages a teleologic rather than causal response.) The answer to this question has the potential to help you 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 higher-level requirements this one satisfies.

Examples:
Unacceptable: Traffic lights shall be used to control traffic at the junction. *{Traffic lights are a solution}* *Acceptable (several requirements):* The Traffic_Control_System shall allow the Pedestrian to cross the road at the junction. *{This is the motivating purpose.}* The Traffic_Control_System shall allow Vehicles travelling east-west to traverse the Junction with a maximum wait of 2 minutes during normal daytime traffic conditions. *{Note that glossary definitions should be used for 'Pedestrian', Vehicle', and 'Junction'.}*

Characteristics that are established by this rule:
C2 - Implementation Independent

5.9.2 R34 - /ABSTRACTION/SOLUTIONVOCAB

If writing in the solution domain, express the system level solution in the vocabulary of the system.

Elaboration:
System requirements should provide a high-level specification of the overall solution. The first level of architecture may be laid out, but subsystems are considered as black-boxes to be elaborated at the next level down.

Examples:
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 requirement is full of solution-biased detail.}* *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, which may be a means*

of automatic detection rather than button pushing.}

Characteristics that are established by this rule:

C2 - Implementation Independent

5.10 QUANTIFIERS

5.10.1 R35 - /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 in a closed set.

Examples:

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

Characteristics that are established by this rule:

C3 - Unambiguous

5.11 TOLERANCE

5.11.1 R36 - /TOLERANCE/VALUERANGE

Define the range of acceptable values associated with quantities.

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 requirement to reflect this.
It helps also 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.

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 at least 30 minutes. *{Now we know the range of acceptable flow performance, and that the 30 minutes is a minimum acceptable performance.}*

Characteristics that are established by this rule:
C7 - Verifiable
C8 - Correct
C12 - Feasible

5.12 QUANTIFICATION

5.12.1 R37 - /QUANTIFICATION/MEASURABLE

Provide specific measurable performance targets.

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.

Examples:
Unacceptable: The system shall use minimum power. *Acceptable:* The system shall limit the power consumption to less than 50% of the current solution. *{This both takes into account the underlying goal - to minimize power consumption - and provides a measurable target.}*

Characteristics that are established by this rule:
C7 - Verifiable
C12 - Feasible

5.12.2 R38 - /QUANTIFICATION/TEMPORALINDEFINITE

Define temporal dependencies explicitly instead of using indefinite temporal keywords.

Elaboration:
Some words and phrases signal non-specific timing, such as 'eventually', and 'at last'. These should be replaced by specific timing constraints.

Examples:
Unacceptable: Continual operation of the pump shall eventually result in the tank being empty. *Acceptable*: The Pump shall remove at least 99% of the Fluid from the Tank in no more than 3 days of continuous operation.

Characteristics that are established by this rule:
C7 - Verifiable

5.13 TRACEABILITY

5.13.1 R39 - /TRACEABILITY/UNIQUEREFERENCE

Provide a unique reference for each requirement.

Elaboration:
Tracing individual requirements is greatly aided by having a unique reference number or identifier for each requirement that does not change over time. More and more, requirement management tools are being developed that store the requirements in a database. These tools need to assign a unique number to each requirement that will never be reused for the project even if the requirement is later deleted.

Examples:
Unacceptable: use of section numbers in a Word document, since inserting a new section changes the numbers throughout the remaining document.

Characteristics that are established by this rule:
A1 - Alignment with need

5.13.2 R40 - /TRACEABILITY/PARENTCHILD

Establish traceability between each parent requirement and the set of child requirements that are derived from it.

Elaboration:
This is about being able to trace a parent requirement down to those child requirements that are derived from it, and vice versa. The ability to trace in this way answers the question "how does the solution achieve this requirement?" The goal is to be able to determine whether or not the set of child requirements is necessary and sufficient to meet the intent of the parent. The parent/child relationship may be many-to-many; i.e. a child requirement may serve to satisfy multiple parents. This is also about being able to trace the allocation of a requirement between levels. In many cases, allocation will coincide with decomposition, in that a child requirement of a level *n* parent is immediately allocated to a part at level *n*+1. However, several stages of decomposition may occur within a level before allocation occurs. When the requirement is allocated to more than one part at the next level, there may be an internal interface involved as show in the example.

Examples:
Acceptable tracing:

System requirement:
The Traffic_Control_System shall allow the Pedestrian to signal intent to cross the road.
Traces to three component requirements that achieve it:
<-- The Pedestrian_Detection_Device shall detect the presence of the Pedestrian on the Pavement beside the Crossing_Point.
<-- The Pedestrian_Detection_Device shall transmit a Pedestrian_Present signal to the Signal_Control_Unit when the presence of a Pedestrian is detected.
<-- The Signal_Control_Unit shall receive a Pedestrian_Present signal from the Pedestrian_Detection_Device.
{Note that the definition of the signals named in these requirements would be given separately in a glossary of data dictionary.
Note also that this example shows the traces as arrows beside the derived child requirements. In practice, a requirements management tool would make use of unique requirement numbers to represent the relationship.}

Characteristics that are established by this rule:
C10 - Complete
A1 - Alignment with need
A2 - Alignment with design

5.13.3 R41 - /TRACEABILITY/VERIFICATION

Every requirement statement should be linked to the verification artifacts that contribute to its verification.

Elaboration:
This is about being able to trace a requirement across to the activities that are planned to provide evidence that the requirement will be, or has been, satisfied. This answers the question "How will I know that this requirement has been (or will be) met?" Verification artifacts include analysis, test, inspection, demonstration or other means of providing evidence for satisfaction fo requirements.

Examples:
Traceability to test artifacts is most commonly achieved through a cross-reference matrix that shows, for each requirement, the test artifact(s) that contribute to its verification or validation.

Characteristics that are established by this rule:
A2 - Alignment with design
A3 - Alignment with evidence

5.13.4 R42 - /TRACEABILITY/EXTERNALREFERENCES

Make references to external documents specific to an identifiable element.

Elaboration:

There are times that a requirement needs to point to another document that defines the actual values to be met. Examples include reference to a standard or to a document that defines an interaction (interface) between your system and another. As an aid to readability, it is best to refer to only the document number and not the title. The full title and version number of the document should be documented in as separate "applicable" documents section. Applicable documents are those that are referenced by a requirement and contain binding information that will be verified. It is helpful if such references point to a specific section number, a figure number or a paragraph number, as this will assist in precise verification.

Examples:

There are a number of acceptable mechanisms that can be used to refer to content in external documents: '...in accordance with ...', '... as specified in ...', or '... as set out in ...'. Perhaps '...in accordance with ...' are the most useful.

Characteristics that are established by this rule:

C8 - Correct

A1 - Alignment with need

A2 - Alignment with design

6 RULES FOR SETS OF REQUIREMENTS

6.1 UNIFORMITY OF LANGUAGE

6.1.1 R43 - /UNIFORMLANGUAGE/IMPORTANCE

Use an agreed convention for distinguishing the relative importance of requirements.

Elaboration:

There are several conventions used for classifying requirements according to their relative priority or importance in meeting the stakeholder expectations. Note that all requirements are binding and will be verified. Knowing the priority allows you to better manage your requirements, focusing efforts on those of higher importance. When problems come up, priorities allow you to make trades. Requirement priorities must be made known to all stakeholders. As requirements flow down from one level to another, the priority of child requirements is influenced their parent's priority.

Examples:

Keep it simple, like 1, 2, & 3. Have rules for each category. FOr instance:
Priority 1 – *essential need to rethink scope if can't have these. Use these to identify where to apply resources when problems/change occurs (best bang for the buck). Never trade off.*
Priority 3 – *flexible or useful can be added later, expand tolerances – negotiable or deliver in a later phase (if doing incremental development). Provides trade space.*
Priority 2 – *all other requirements. Trade off if needed.*

Characteristics that are established by this rule:
C9 - Conforming
A3 - Alignment with evidence
A4 - Priority

6.1.2 R44 - /UniformLanguage/ConsistentTerms

Use a glossary to define a consistent set of terms to be used in requirement statements.

Elaboration:
Have and use just one term per concept in the whole set of requirements. Synonyms are not acceptable. Use a glossary to define each term used. This may be as simple as a list of nouns and verbs allowable in the project.

Examples:
Unacceptable: It would not be acceptable for one requirement to refer to the system using one term and another to refer to the same system using another term. *Acceptable: Settle on only one term, define it in the glossary, and then use it consistently in each requirement.*

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

6.1.3 R45 - /UniformLanguage/Acronyms

Use an acronym list to define a consistent set of acronyms to be used in requirement statements.

Elaboration:
As with the glossary, the same acronym must be used in each 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 use can lead to ambiguity.

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 items 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.*

Characteristics that are established by this rule:
C9 - Conforming

C11 - Consistent
C13 - Bounded

6.1.4 R46 - /UniformLanguage/Abbreviations

Use an abbreviation list to define a consistent set of abbreviations to be used in requirement statements.

Elaboration:
As with the glossary and acronyms, the same abbreviation must be used in each requirement; multiple meanings of the same abbreviation are not acceptable. Inconsistency in use can lead to ambiguity.

Examples:
Unacceptable: It would not be acceptable for one requirement to refer to the 'op' meaning operation and another to refer to the 'op' meaning the operator. *Acceptable: Settle on only one abbreviation, define it in the list of abbreviations, and then use it consistently in each requirement.*

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

6.1.5 R47 - /UniformLanguage/StyleGuide

Use a project-wide style guide.

Elaboration:
A style guide is important to ensure that requirements are written in a consistent manner, which aids in achieving unambiguous requirement sets.

Examples:
For example, each organization should have a style guide to address such issues as the template for statement, the use of priority, standard abbreviations and acronyms, layout and use of figures and tables, layout of requirements documents, requirement numbering, and use of databases.

Characteristics that are established by this rule:
C5 - Singular
C9 - Conforming
C11 - Consistent
C13 - Bounded
C14 - Structured
A3 - Alignment with evidence

6.2 MODULARITY

6.2.1 R48 - /MODULARITY/DEPENDENTS

Group mutually dependent requirements together.

Elaboration:
This will often mean that requirements are grouped by feature, bringing a functional requirement together with a number of related constraints and performance requirements.

Examples:
Performance requirements should be grouped with their related functional requirement.

Characteristics that are established by this rule:
C14 - Structured

6.2.2 R49 - /MODULARITY/RELATEDREQUIREMENTS

Group related requirements together.

Elaboration:
Requirements that belong together should live together. This is a good principle for structuring a requirements document.

Examples:
Requirements may be related by:- *- type (e.g. all safety requirements);* *- scenario (e.g. requirements arising from a single scenario).*

Characteristics that are established by this rule:
C14 - Structured

7 REFERENCE DOCUMENTS

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

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

ISO/IEC, *ISO/IEC 29148 FDIS Systems and Software Engineering—Life Cycle Processes—Requirements Engineering, 2011*.

APPENDIX A. ACRONYMS AND ABBREVIATIONS

AFIS: Association Française d'Ingénierie Systèmes
IEC : International Electrotechnical Commission
INCOSE : International Council on Systems Engineering
ISO : International Standards Organisation
REGAL : Requirements Engineering Guide for All
RWG: Requirements Working Group
SysML: Systems Modeling Language
Vocab: Vocabulary

APPENDIX B. COMMENT FORM

Reviewed document:	Insert Document Title
Name of submitter (first name & last name):	Firstname Lastname
Date Submitted:	21-Aug-2010
Contact info (email address)	john.doe@anywhere.com
Type of submission (individual/ group):	IndividualGroup
Group name and number of contributors (if applic.):	INCOSE XYZ WG

Please read examples carefully before providing your comments (and delete the examples provided.)

Comment sequence no.	Commenter name	Category (TH, TL, E, G)	Section number (eg, 2.1.1, no alpha)	Specific reference (e.g., paragraph, line, Figure #, Table #)	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.

(BACK COVER)