Published in the *Proceedings of the Third International Symposium of the NCOSE - Volume 2, 1993*.
Prepared by the Requirements Working Group of the International Council on Systems Engineering, for information purposes only. Not an official position of INCOSE.

Writing Good Requirements

(A Requirements Working Group Information Report)

Ivy Hooks
Compliance Automation, Inc.
17629 Camino Real, Suite 207
Houston, Texas 77058

Abstract. The primary reason that people write poor requirements is that they have had no training or experience in writing good requirements. This paper will address what makes a good requirement. It will cover some of the most common problems that are encountered in writing requirements and then describe how to avoid them. It also includes examples of problem requirements and how to correct them.

INTRODUCTION

If you or your staff are having problems with writing good requirements, you may benefit from guidance in how to write good requirements. The college courses you took probably never mentioned the subject of requirements. Even if you have taken classes in system engineering or program management, you may have had only an introduction to the subject of writing requirements. If you are using existing specifications for guidance, you may be using poor examples. The information provided in this paper is used in training people to write good requirements and generally results in a step function improvement in the quality of requirements.

An important aspect of system engineering is converting user "needs" into clear, concise, and verifiable system requirements. While this paper primarily discusses system level requirements, it is equally applicable at all lower levels.

The first section discusses what is a good requirement. This is followed by a discussion of common problems and how to avoid them. It also includes examples of problem requirements and how to correct them.

GOOD REQUIREMENTS

A good requirement states something that is **necessary**, **verifiable**, and **attainable**. Even if it is verifiable and attainable, and eloquently written, if it is not necessary, it is a good requirement. To be verifiable, the requirement must state something that can be verified by examination, analysis, test, or demonstration. Statements that are subjective, or that contain subjective words, such as "easy", are not verifiable. If a requirement is not attainable, there is little point in writing it. A god requirement should be clearly stated.

Need. If there is a doubt about the necessity of a requirement, then ask: *What is the worst thing that could happen if this requirement were not included?* If you do not find an answer of any consequence, then you probably do not need the requirement.

Verification. As you write a requirement, determine how you will verify it. Determine the criteria for acceptance. This step will help insure that the requirement is verifiable.

Attainable. To be attainable, the requirement must be technically feasible and fit within budget, schedule, and other constraints. If you are uncertain about whether a requirement is technically feasible, then you will need to conduct the research or studies to determine its feasibility. If still uncertain, then you may need to state what you want as a goal, not as a requirement. Even is a requirement is technically feasible, it may not be attainable due to budget, schedule, or other, e.g., weight, constraints. There is no point in writing a requirement for something you cannot afford -- be reasonable.

Clarity. Each requirement should express a single thought, be concise, and simple. It is important that the requirement not be misunderstood -- it must be unambiguous. Simple sentences will most often suffice for a good requirement.

COMMON PROBLEMS

The following is a list of the most common problems in writing requirements:

- **Making bad assumptions**
- **Writing implementation (HOW) instead of requirements (WHAT)**
- **Describing operations instead of writing requirements**
- **Using incorrect terms**
- **Using incorrect sentence structure or bad grammar**
- **Missing requirements**
- **Over-specifying**

Each of these are discussed in detail below.

BAD ASSUMPTIONS

Bad assumptions occur either because requirement authors do not have access to sufficient information or the information does not exist. You can eliminate the first problem by documenting the information critical to the program or project, including:

- Needs
- Goals
- Objectives
- Constraints
- Missions
- Operations Concept
- Budget
- Schedule
- Management/Organization

This information then must be made available to all authors. You can create and maintain a list of other relevant documents and make these easily accessible to each author. If you have automated the process, you can offer documents on-line and you can filter the information within the documents so that individual authors can get copies of only the data that they need.

In the second case where information does not exist, the requirement author should document all assumptions with the requirement. When the requirement is reviewed, the assumptions can also be reviewed and problems quickly identified. It is also useful to document the assumptions even if the authors were provided the correct information. You cannot ensure that all authors have read all the information or interpreted it correctly. If they document their assumptions, you will avoid surprises later.

IMPLEMENTATION

A specification was released for the development of a requirements management tool. The first requirement was to "provide a data base". The statement is one of implementation and not of need, and it is common to find such statements in requirement specifications. Specifications should state *WHAT* is needed, not *HOW* it is to be provided. Yet this is a common mistake made by requirement writers. Most authors do not intend to state implementation, they simply do not know how to state the need correctly.

To avoid stating implementation, ask yourself *WHY* you need the requirement. In the example cited, it can be seen that by asking *WHY*, the author can define all of the needs that the system must meet and will then state the real requirements, e.g.:

- *provide the capability for traceability between requirements*

- *provide the capability to add attributes to requirements*
- *provide the ability to sort requirements*

These requirements state *WHAT* is needed, not *HOW* to accomplish it. Each of the above listed requirements might result in a data base type of system, but the requirement for the data base was not needed.

There are two major dangers in stating implementation. The one most often cited is that of forcing a design when not intended. If all the needs can be met without a data base, then why state the need for a data base. If they cannot be met, another, or better way, then a data base will be the solution -- whether or not there was a requirement for a data base.

The second danger is more subtle and potentially much more detrimental. By stating implementation, the author may be lulled into believing that all requirements are covered. In fact, very important requirements may be missing, and the provider can deliver what as asked for and still not deliver what is wanted. Providing a data base will not be sufficient for someone needing a requirements management tool that need to be stated as requirements.

At each level of requirements development the problem of needs versus implementation will occur. At the system level the requirements must state WHAT is needed. The system designer will determine HOW this can be accomplished and then must define *WHAT* is needed at the subsystem level. The subsystem designer will determine *HOW* the need can be met and then must define *WHAT* is needed at the component level.

To ensure that you have not stated implementation, ask yourself *WHY* you need the requirement. If this does not take you back to a real need statement, then you are probably stating a need and not implementation.

The Implementation Trap. If you have been doing design studies at a low level, you may begin to document these results as high level requirements -- this is the implementation trap. You will be defining implementation instead of requirements.

An example of this occurred during the definition of the Assured Crew Return Vehicle (ACRV) requirements. An individual submitted a requirement like this:

- *The ACRV System shall enter when sea state is at TBD conditions.*

The ACRV had no requirement for a water landing -- that was a design option. The individual had been working with that design option, and from previous Apollo experience known that crew rescue was possible only in certain sea sates.

When asked WHY the requirement was needed, the individual stated that the crew could not be left in the module for a lengthy period of time, thus the landing needed to be where and when sea states could accommodate crew rescue. He had a valid requirement -- but

not the one he had written. Whether the ACRV landed on water or land, removing the crew within a limited time period was essential. Thus the real requirement was:

- *The ACRV System shall provide for crew removal within TBD time of landing.*

The question *WHY* will resolve most implementation requirement errors. Always ask *WHY* a requirement is needed to insure that you have not fallen into this lower level requirement trap.

OPERATIONS VS. REQUIREMENTS

This problem is somewhat similar to the implementation problem. The Simplified Aid for EVA Rescue (SAFER) project provides examples of this problem.

The author intended to write an environment requirement in the SAFER Flight Test Article (FTA) Specification. The statement was written as a description of operations, not a requirement about the environment.

- *The SAFER FTA shall be stowed in the Orbiter Airlock Stowage Bag for launch landing, and on-orbit stowage.*

The real requirement is:

- *The SAFER FTA shall be designed for the stowage environment of the Airlock Storage Bag for launch, entry, landing, and on-orbit, as defined in TBD.*

The next requirement again describes the operations and is confusing.

- *The SAFER FTA shall be operated by an EVA Crew member wearing EMU sizes small through extra large without limiting suit mobility.*

The statement was rewritten and resulted in a requirement and a design goal. The design goal is needed because no quantifiable requirement can be written regarding suit mobility.

- *The SAFER FTA shall be designed for use with EMU sizes small through extra large.*
- *The SAFER FTA should not limit EVA crew member mobility.*

The danger in stating operations, instead of a requirement is (1) the intent may be misunderstood and (2) determining how to verify can be a problem.

USE OF TERMS

In a specification, there are terms to be avoided and terms that must be used in a very specific manner. Authors need to understand the use of ***shall***, ***will***, and ***should***:

- Requirements use *shall*.
- Statements of fact use *will*.
- Goals use *should*.

These are standard usage of these terms in government agencies and in industry. You will confuse everyone if you deviate from them. All *shall* statement (requirements) must be verifiable, otherwise, compliance cannot be demonstrated.

Terms such as are, *is, was,* and *must* do not belong in a requirement. They may be used in a descriptive section or in the lead-in to a requirements section of the specification.

There are a number of terms to be avoided in writing requirements, because they confuse the issue and can cost you money, e.g.

- Support
- But not limited to
- Etc.
- And/Or

The word support is often used incorrectly in requirements. Support is a proper term if you want a structure to support 50 pounds weight. It is incorrect if you are stating that the system will support certain activities.

- ***WRONG:*** *The system shall support the training coordinator in defining training scenarios.*

- ***RIGHT:*** *The system shall provide input screens for defining training scenarios. The system shall provide automated training scenario processes.*

The terms ***but not limited to***, and ***Etc.*** are put in place because the person writing the requirements suspects that more may be needed than is currently listed. Using these terms will not accomplish what the author wants and can backfire.

The reason the terms are used is to cover the unknown. The contractor will not increase the cost in the proposal because you added these terms. The only way to get the work added is to place an analysis task in the Statement of Work (SOW) to determine if more items need to be added to the list. In the SOW you can control what effort the contractor will expend to address these unknowns. If more items are found, you may have to increase the scope of the contract to cover the additions.

If you have these terms in your requirements specification, the contractor may use them as an excuse for doing unnecessary work for which you must pay. You cannot win by using the terms in the specification.

The term ***and/or*** is not appropriate in a specification. If you use and/or and the contractor does the or he has met the terms of the contract. Either you want item 1 and item 2 or you will be satisfied with item 1 or item 2. Again, if you use the term or, then the contractor has met the terms of the contract if he does either item.

REQUIREMENT STRUCTURE/GRAMMAR

Requirements should be easy to read and understand. The requirements in a system specification are either for the system or its next level, e.g. subsystem. Each requirement can usually be written in the format:

- *The System shall provide .....*
- *The System shall be capable of .....*
- *The System shall weigh .....*
- *The Subsystem #1 shall provide .....*
- *The Subsystem #2 shall interface with .....*

Note: The name of your system and the name of each subsystem appears in these locations. If you have a complex name, please use the acronym, or your document will have many unneeded pages just because you have typed out a long name many times.

Each of these beginnings is followed by *WHAT* the System or Subsystem shall do. Each should generally be followed by a single predicate, not by a list. There are situations where a list is appropriate, but lists are over-used. Since each item in the list must be verified, unless all items will be verified by the same method and at the same time, it is generally not appropriate to put items in a list.

Requirement statements should not be complicated by explanations of operations, design, or other related information. This non-requirement information should be provided in an introduction to a set of requirements or in rationale.

You can accomplish two things by rigorously sticking to this format. First, you avoid the Subject Trap. Second, you will avoid the Subject Trap. Second, you will avoid bad grammar that creeps into requirements when authors get creative in their writing.

Subject Trap. A system specification contains requirements for the system, e.g., you may want to require that the system provide control. A set of requirements might be written that reads as follows:

- *The guidance and control subsystem shall provide control in six degrees of freedom.*
- *The guidance and control subsystem shall control attitude to 2 0.2 degrees.*
- *The guidance and control subsystem shall control rates to 0.5 0.05 degrees/second.*

The subject trap is created because the author has defined a guidance and control subsystem. Controlling attitude and rate is a system problem, it requires not only a guidance and control subsystem but also a propulsion subsystem to achieve these attitudes and rates. Determining what subsystems will be required to accomplish the requirements is part of the design process The requirements should be written from the system perspective, as follows:

- *The system shall provide six degrees of freedom control.*
- *The system shall control attitude to 2 0.2 degrees.*
- *The system shall control rates to 0.5 0.05 degrees/second.*

The author of the original requirements was not trying to define the lower level breakout. He probably comes from a control background and sees the system from that perspective and hence writes requirements that way. The flow down of requirements, to all affected segments, elements, and subsystem, will be badly affected if these requirements are not written correctly.

Bad Grammar. If you use bad grammar you risk that the reader will misinterpret what is stated. If you use the requirements structure suggested above, you will eliminate the bad grammar problems that occur when authors try to write complex sentences and use too many clauses.

Another solution is to write requirements as bullet charts. When the content is agreed upon a good writer can convert the information into a sentence for the specification.

Authors will also try to put all that they know in a single sentence. This results in a long complex sentence that probably contains more than one requirement. Bullet charts or one good editor can alleviate this problem. The following requirement contains multiple requirements.

- *The ACRV System shall provide special medical life-support accommodations for one ill or injured crew member consisting of medical life-support and monitoring equipment and the capability of limiting impact accelerations on that crew member to be not greater than.... for a total impulse not to exceed ....*

The requirement needs to be broken into at least four requirements and it could use a lead-in such as:

- *The ACRV will be used as an ambulance for an ill or injured crew member. Only one crew member will accommodate at a time. The following define the unique requirements for this capability.*
    - *..provide medical life-support accommo-dations for one crew member*
    - *..provide monitoring equipment for one crew member*
    - *..limit impact accelerations to the ill or injured crew member to no greater than...*
    - *..limit total impulse to the ill or injured crew member to ...*

UNVERIFIABLE

- ***Every requirement must be verified***.

Because every requirement must be verified, it is important to address verification when writing the requirements. Requirements may be unverifiable for a number of reasons. The following discusses the most common reason -- use of ambiguous terms.

Ambiguous Term. A major cause of unverifiable requirements is the use of ambiguous terms. The terms are ambiguous because they are subjective -- they mean something different to everyone who reads them. This can be avoided by giving people words to avoid. The following lists ambiguous words that we have encountered.

- Minimize
- Maximize
- Rapid
- User-friendly
- Easy
- Sufficient
- Adequate
- Quick

The words ***maximize*** and ***minimize*** cannot be verified, you cannot ever tell if you got there. The words minimum and maximum may be used is the context in which they are used can be verified.

What is ***user-friendly*** and what is ***rapid***? These may mean one thing to the user or customer and something entirely different to a designer. When you first begin writing your requirements, this may be what you are thinking, but you must write the requirements in terms that can be verified. If you must use an ambiguous term in first draft documents, put asterisks on either side of the term to remind yourself that you are going to have to put something concrete in the requirement before you baseline the document.

There may be cases where you cannot define, at your level, exactly what is needed. If this is the case, then you should probably be writing a design goal, not a requirement. You can do this by clearly indicating that your statement is a goal, not a requirement. Use of the word should, instead of the shall, will denote a design goal.

MISSING

Missing items can be avoided by using a standard outline for your specification, such as those shown in Mil-Std-490 or IEEE P1233, and expanding the outline for your program.

Many requirements are missed because the team writing the requirements is focused on only one part of the system. If the project is to develop a payload, the writers will focus on the payload's functional and performance requirements and perhaps skip other important, but less obvious, requirements. The following is a checklist of requirement drivers you need to consider:

- Functional
- Performance
- Interface
- Environment
- Facility
- Transportation
- Deployment
- Training
- Personnel
- Reliability
- Maintainability
- Operability
- Safety
- Regulatory
- Security
- Privacy
- Design constraints

You will need to develop detailed outlines for your specification for the functional and performance requirements, and in perhaps other areas.

You may also have a number of requirements that you must include by reference. In particular, those standards that define quality in different disciplines (materials and processes) or for different projects.

Detailed requirements analysis is necessary to assure that all requirements are covered. There are a number of approaches to performing requirements analysis and a number of tools for doing this work. Detailed requirements analysis is beyond the scope of this paper.

OVER-SPECIFYING

The DoD has stated that over-specification is the primary cause of cost overruns on their programs. Over-specifying is most often from stating something that is unnecessary or from stating overly stringent requirements.

Unnecessary Items. Unnecessary requirements creep into a specification in a number of ways. The only cure is careful management review and control.

People asked to write requirements will write down everything they can think of. If you do not carefully review each requirement and why it is needed before baselining the specification, the result will be a number of unneeded requirements.

Example: The Space Station Training Facility (SSTF) had a requirement for a high-fidelity star field. The author knew that a new high-fidelity star field was being developed for the Shuttle Mission Simulator (SMS) and assumed they might as well put the same thing in the SSTF. The crew needs a background to view outside the Space Station, but there is no need for a high-fidelity star field, since they do not use the stars for navigation.

The requirement needs to be written for a visual background for crew orientation. The design process will determine if using the SMS star field is a cost effective solution or it something simpler is adequate and more cost effective.

Example: A number of SSTF requirements were deleted when their authors were queried as to the need. The need they stated was that "it would be nice to have". Most program do not have budgets for *nice to have items.*

Unnecessary requirements can also appear after baseline if you let down your review and control process. In ACRV a number of requirements were added after the initial baseline that were not needed. One such instance occurred because of an error in the baseline document.

Example: The baseline document had two requirements:

- *The ACRV System shall be capable of operating over a planned operational life of thirty (30) years.*
- *The Flight Segment shall provide an operational life of 30-years for the flight elements.*

The second requirement, for the Flight Segment, was not required since the System requirement was adequate. The action that should have been taken was to delete the Flight Segment requirement. Instead, two more requirements were added to require a 30-year operational life of the other two segments.

At least one other requirement was added to ACRV that was a duplicate of an existing requirement. The wording of the two differed only slightly and their rationale was the same. It requires careful attention to detail to avoid this type of problem.

Over Stringent. Most requirements that are too stringent are that way accidentally, not intentionally. A common cause is when an author writes down a number but does not consider the tolerances that are allowable.

Thus, you should not state that something must be a certain size, e.g., 100 ft.2, if it could just as easily be 100 10 ft.2. You do not need to ask that something deliver a payload to exactly 200 n.m. if greater than or equal to 200 n.m. is acceptable.

Some of the major horror stories of the aerospace industry deal with overly-stringent requirements. One contractor was severely criticized for charging $25,000 per coffee pot in airplanes built for the government. But the requirements for the coffee pot were so stringent, that the plane could have crashed and yet no coffee could spill. It cost a great deal to develop the coffee pot and to verify that it met its requirements. Each copy had to be built to a stringent design.

The solution to this problem is to discuss the tolerances allowable for any value and then to write the requirement to take into consideration those tolerances. Each requirement's cost should also be considered.