June 13, 2016 – QRA Team

21 Top Engineering Tips for Writing an Exceptionally Clear Requirements Document

Because nobody likes building or using a poor requirements document. Over the past year, our team has probed dozens of engineers and their requirements documents to create the ultimate list of tips on how to write requirements documents that are a dream to work with.

Because nobody likes building or using a poor requirements document.

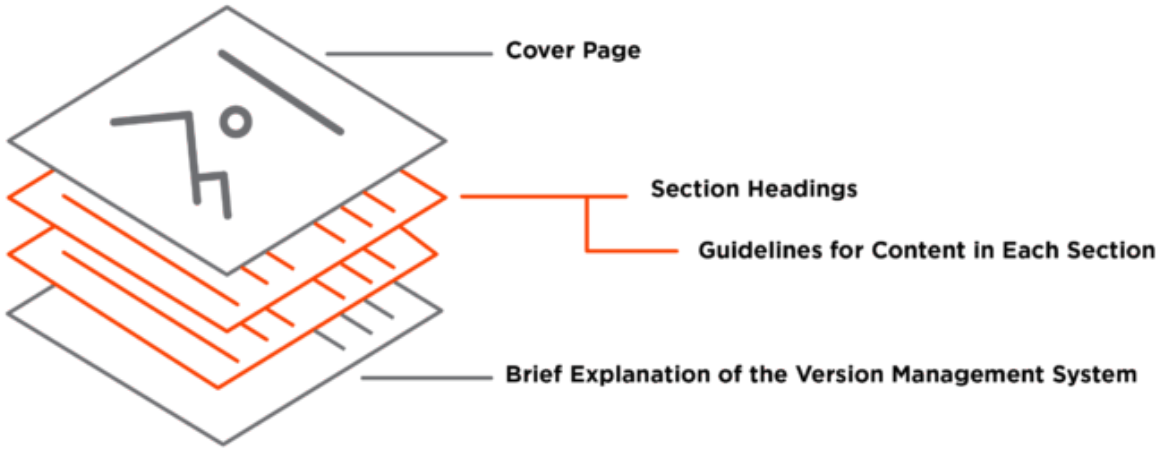
It has become clear that enormous numbers of engineering design errors originate in the requirements document. And agreement on requirements engineering best practices is fiercely debated. Everyone has their own opinions, which differ widely. We've distilled the information from our research and interviews into this one insight-packed guide that we hope will settle some debates.

Table of Contents

1. Use a (Good) Requirements Document Template

Every requirements engineer we interviewed uses a template when starting a new requirements document. If you don't, you should. And if you do, you should make sure your template is a good one.

A good requirements document template should have at minimum a cover page, section headings, essential guidelines for the content in each section and a brief explanation of the version (change) management system used to control changes made to the document.



Your template should also include standardized sections covering topics like verb (imperative) application, formatting and traceability standards, and other guidelines your organization follows in documenting requirements and managing its requirements documentation.

Standardized sections – or "boilerplate" as they are often called – promote and facilitate consistency across projects. This is a major benefit of templates. These sections tend to remain little changed from project to project, and from team to team within a company – evolving only slowly over time with changes in methodology and lessons learned – thus providing a stable platform for consistent requirements development, employee education and communication with customers.

2. Organize in a Hierarchical Structure

To deliver a document that is easy to use from top to bottom, organize your requirements in a hierarchical structure. Hierarchical structures can include manager–supplier, function–sub-function, mission–part, etc.

A common 3 tier hierarchy system for a Mission-level requirements document might look something like this:

Level	Example
Mission	On-orbit, Highway, Moor
System	Spacecraft ops, Truck ops, Vessel ops
Phase	De-orbitting, Cruise Control, Docking

This method of organization helps you focus on each specific domain that needs to be addressed, and thus author requirements documents that are as comprehensive as possible. It also helps you easily find the areas you need to modify in the baseline specification when adding functionality to an existing system. Last, but not least, it allows requirements users to quickly drill down to the exact functional area they are looking for.

Many organizations will begin their requirements documents at the subsystem or component level depending on the nature of their business. A hierarchical structure should still be used.

In component specifications, for example, a functional hierarchy is often used, with very broad functional missions at the top breaking down into sub-functions, and those sub-functions breaking down into successive tiers of sub-functions.

3. Use Identifiers to Your Advantage

It may come as a surprise, but many requirements documents lack a comprehensive requirement identification system.

Requirement identifiers are often a requirement themselves. Systems purchased under contract between a customer and a supplier – as in the case of most government-purchased systems, for example – are normally developed in accordance with an industry-accepted standard, like IEEE/EIA 12207, as a stipulation of the contract. Such standards typically require that each requirement in every requirement document be tagged with a *project unique identifier* (PUI).

And for good reason.

Tagging each requirement with a PUI improves and simplifies traceability between high-level and low-level requirements, and between requirements and verification tests. Brief identifiers make it easy to build traceability tables that clearly link each requirement to its ancestors in higher level documents, and to the specific tests intended to verify it. Traceability tables simplify the process of demonstrating to the customer and internal stakeholders that the system has been developed to, and proven to comply with, the agreed top-level requirements.

What's more, linking these unique identifiers to the hierarchical structure of your requirements document – in other words, basing your PUIs on the paragraph numbers of the document – makes it easy for users to find referenced requirements within the document itself.

Requirements documents that do not employ such an identifier system are not only difficult to read and reference, they make traceability a nightmare.

Therefore, each requirement should be marked with a PUI that allows users to easily reference both the requirement and its position in the overall document.

Let's look at an example. NASA's ISS Crew Transportation and Services Requirements Document contains the following requirement 3.5.2.5:

3.5.2.5 Spacecraft Ventilation for Emergency Landings The spacecraft shall provide cabin ventilation equivalent to 4 cabin air exchanges per crewmember per hour while crew is present after an emergency landing. [R.CTS.364] *Rationale: A remote landing could subject the spacecraft and crew to harsh environmental conditions ranging from high atmospheric temperatures to rough seas. If the crew must remain in the vehicle, this ventilation will equalize cabin temperature, mitigate CO2 buildup, and replenish O2. The duration of this service and the variability of ventilation rates with landing environments are developed in conjunction with the crew survivability strategy in requirement 3.5.2.4.* The PUI for this requirement – 3.5.2.5 – indicates the exact position in the document in which this requirement is stated, according to the following section/subsection/paragraph hierarchy:

3: ISS Crew Transportation and Service Requirements

5: Entry/Landing Requirements

2: Contingency

5: Spacecraft ventilation for emergency landings

Also note that this identification system allows NASA to also link requirements to related requirements – in this case requirement 3.5.2.4: Crew Survival after Emergency Landing – by referencing them in rationale statements (see Tip #8, below).

4. Standardize Your Requirements Document Language

Like most spoken languages, English is full of words that have multiple definitions and which evoke subtle shades of meaning. This is a great thing when it comes to self-expression, but can lead to confusion and disagreement when it comes to specifying and interpreting requirements.

A good tactic for reducing ill-definition and misinterpretation of requirements is to standardize the language you are going to use to express them. A good way to do this is with a dedicated section toward the beginning of your requirements document (part of your template). This section will define exactly how certain terms will be used within the document itself, and how they should be interpreted when found in non-requirements documents referenced by the document.

The following segment is a good example of language standardization from NASA's ISS Crew Transportation and Services Requirement Document:

When used within the context of a requirement under a contract, statements in this document containing shall are used for binding requirements that must be verified and have an accompanying method of verification; will is used as a statement of fact, declaration of purpose, or expected occurrence; and should denotes an attribute or best practice which must be addressed by the system design. When used within the context of a reference document under an agreement, the verbs shall, will, and should are only intended as informational and are not binding.In some cases, the values of quantities included in this document have not been confirmed and are designated as: "To Be Confirmed" (TBC) – still under evaluation, and "To Be Determined" (TBD) or "To Be Supplied" (TBS) – known, but not yet available. A "To Be Resolved" (TBR) is used when there is a disagreement on the requirement between technical teams. When a change in a noted characteristic is deemed appropriate, notification of the change shall be sent to the appropriate review and change control authority.

Each requirement in CCT-REQ-1130 is annotated by its section number. At the end of each requirement text is a requirement ID of the format R.CTS. This corresponds to the absolute ID in NASA's requirements database. It can be used to cross reference requirements in this document to spreadsheet exports of the database. See Section 1.3 in the event of conflict between this document and spreadsheet exports.

Strictly defining your terms and adhering strictly to your definitions will not only reduce conflict and confusion in interpreting your requirements – with practice, using standardized language will also save you time in writing requirements.

5. Be Consistent with Imperatives

One of requirements engineering's greatest debates is on the use of *imperatives*, words like shall, must, will, should, etc…

Although there were some dissenters amongst the requirements engineers we interviewed, the consensus was to crown "shall" as a binding provision. Non-binding provisions are indicated by the word "should" or "may." And a declaration of purpose is indicated by the word "will."

Also, many requirements engineers like to use the word "must" to express constraints and certain quality and performance requirements (non-functional requirements). The use of "must" allows them to express constraints without resorting to passive voice (see Tip #14), and to easily distinguish between functional requirements (expressed with "shall") and non-functional requirements (expressed with "must").

Once you have agreement on how each imperative term will be used within your organization, document that agreed usage within your requirements document template (as illustrated in Tip #4).

In general the rules for using imperatives are simple. Use exactly one provision or declaration of purpose (such as shall) for each requirement, and use it consistently across all requirements.

6. Make Sure Each Requirement is Testable.

“Each requirement shall be assigned a project-unique identifier to support testing and traceability and shall be stated in such a way that an objective test can be defined for it.”

Software Requirements Specification (SRS) Data Item Description (DID), MIL-STD-498.

Since appearing in the referenced standard over 20 years ago, that requirement has appeared in a number of subsequent standards and in scores of requirements documents and templates. Yet, it’s surprising how many requirements – *written under those same standards* – fail to meet the second half of that requirement.

Every time you write a new requirement, you must ask yourself,

“How will successful implementation of this requirement be verified?”

Writing your requirement with a specific test scenario in mind will help ensure that both design and test engineers understand exactly what they have to do.

Of course, the nature of the test scenario – the manner in which the requirement will be verified – will influence how narrowly the requirement has to be defined. Higher level requirements are often tested by inspection or through user testing (flight testing, test driving, etc.) and thus may be quite broad in scope. Lower level requirements that will be verified through software testing or system integration testing must normally be specified to a finer degree of detail.

A good practice for insuring requirement testability, for example, is to specify a reaction time window for any output event the software must produce in response to a given input condition, as in the following example:

3.8.5.3.1: The Engine Monitor shall set <Overtemp Alert> to TRUE within 0.5 seconds when <Engine Temp> equals or exceeds 215° F.

7. Write Functional Requirements to be Implementation-Neutral

What does “implementation-neutral” mean? It means that functional requirements should not restrict design engineers to a particular implementation. In other words, functional requirements should be free of design details.

Writing functional requirements in an implementation-neutral manner has a number of **benefits**:

- Allows design engineers to design the system in the most efficient manner available.
- Allows implementation to be modified without affecting (rewriting) the requirement, as long as the requirement can still be fulfilled by the new implementation.
- Greatly reduces the possibility of conflict between (and rewriting of) requirements due to incompatibility of implementation details.

A good way to avoid dictating implementation is to write your functional requirements strictly in terms of the external interface or externally observable behaviour of the system being specified. That means functional requirements should specify the required external output behaviour of the system for a stated set or sequence of inputs applied to its external interfaces.

In other words, state ***what*** the system must do, ***not how*** it must do it.

Constraints on manner of implementation should not appear in functional requirements. They should be spelled out in very specific *non-functional* requirements at the outset of the program.

8. Rationale Statements are Always Appreciated

Rationale statements are another great tool for reducing ambiguity in your requirements document. They allow you to simplify your requirements statement while providing users with additional information.

A short and concise sentence is usually all that is needed to convey a single requirement – but it's often not enough to justify a requirement. Separating your requirements from their explanations and justifications enables faster comprehension, and makes your reasoning more evident.

The following requirement from NASA's ISS Crew Transportation and Services Requirement Document is a great example of a rationale statement's utility.

3.8.5.1.5 Operable by Single Crewmember The spacecraft shall be operable by a single crewmember for operations that require crew control. [R.CTS.135]

Rationale: The vehicle must be designed so that mission events can be completed by a single crewmember. In addition, vehicle design for single crewmember operations drives operations simplicity and contributes to operations affordability. This requirement results from lessons learned from the Shuttle cockpit, which had critical switches that are out of the operator's reach zone and software that requires more than one crewmember to perform a nominal operation. This requirement does not preclude provision of multiple crew stations for backup and crew resource management (CRM) operations.
The requirement itself is very short and straightforward. The rationale statement supplements it by stating some of the factors (simplicity and affordability) that drove the inclusion of the requirement, and the history behind those driving factors (lessons learned from operation of the earlier Shuttle cockpit). It also states a caveat (does not preclude multiple crew stations) to preempt misinterpretation of the requirement's boundaries.

When a requirement's rationale is visibly and clearly stated, its defects and shortcomings can be more easily spotted, and the rationale behind the requirement will not be forgotten. Rationale statements also reduce the risk of rework, as the reasoning behind the decision is fully documented and thus less likely to be re-rationalized… as so often happens!

When creating a rationale statement, begin by asking yourself the following questions:

- What is an aspect of this requirement that could be a source of contention?
- How am I choosing to address that aspect in the requirement?
- What is the evidence to support my decision?
- What other requirements might have some effect on the interpretation and implementation of the requirement and thus should be referenced in the rationale?

9. Remember that Directives are there to Help You

One of the most underused tactics in requirements writing is the use of *directives*.

Directives are words or phrases that point to additional information which is external to the requirement, but which clarifies the requirement. Directives typically employ phrases like "*as shown in*" and "*in accordance with,*" and they often point to tables, illustrations or diagrams. They may also reference other requirements or information located elsewhere in the document.

The following requirement from NASA's ISS Crew Transportation and Services Requirement Document is a great example of use of a directive:

3.2.5.4 Emergency Lighting The CTS shall provide automatically activated emergency lighting for crew egress and operational recovery in accordance with Table 3.2.5.4-1. [R.CTS.044]

Rationale: Emergency lighting is a part of the overall lighting system for all vehicles. It allows for crew egress and operational recovery in the event of a general power failure. Efficient transit includes appropriate orientation with respect to doorways and hatches, as well as obstacle avoidance along the egress path. The emergency lighting system may include unpowered illumination sources that provide markers or orientation cues for crew egress. Design guidance for emergency lighting can be found in NASA/SP-2010-3407, Human Integration Design Handbook (HIDH).

Table 3.2.5.4-1: Emergency Lighting Intensity Levels

Area(1) or Task(1)	Lux(2)	Ft. C(2)
Passageway	10	1
Emergency Task	32	3

Notes:

(1) Levels are measured at the task object or 789 mm (30 in.) above floor, as applicable.

(2) All levels are minimum.

In this example, the directive is the phrase "in accordance with Table 3.2.5.4-1." Note that while the table is *separate from the requirement statement*, it provides information which clarifies the requirement and thus is an *integral part of the requirement*.

It is vitally important to separate the supporting information referenced by the directive from the requirement statement. Trying to weave complex supporting data into a requirement statement can make the statement overly complex and unclear to the reader. Document users should never have to dig in a haystack to find a clear and specific requirement.

10. Follow Requirement Formatting Best Practices

A key attribute of clear, effective requirements is that they are *concise*. A good technique for authoring concise requirements is to use accepted requirement sentence formats wherever possible.

Engineers who want to write crystal clear requirements would be wise to learn a few basic requirement sentence structures they can apply consistently. A very basic format to start off with is:

Unique ID: Object + Provision/Imperative (shall) + Action + Condition + [optional] Declaration Of Purpose/Expected Occurrence (will)

An example of this format in action is the following:

3.1.5.3 ISS Fly-aroundThe spacecraft shall perform one complete fly-around at a range of less than 250 meters, as measured from spacecraft center of mass to ISS center of mass, after undocking from the ISS.

Unique ID: 3.1.5.3

Object: The spacecraft

Provision: shall

Action: perform one complete fly-around at a range of less than 250 meters, as measured from spacecraft center of mass to ISS center of mass

Condition: after undocking from the ISS
Keep requirements tight. Keep them consistent. And remember: you have rationale (Tip #7) and directives (Tip #8) at your disposal to keep them uncluttered.

11. Use Your EARS to Write Concise Requirements.

We admit it. This is actually a continuation of the previous tip. But we want to give credit where credit is due.

EARS: The Easy Approach to Requirements Syntax developed by Mavin et al. provides a number of proven patterns for writing specific types of requirements.

Requirement Type	Syntax Pattern
Ubiquitous	The <system name> shall <system response>
Event-Driven	WHEN <trigger> <optional precondition> the <system name> shall <system response>
Unwanted Behaviour	IF <unwanted condition or event>, THEN the <system name> shall <system response>
State-Driven	WHILE <system state>, the <system name> shall <system response>
Optional Feature	WHERE <feature is included>, the <system name> shall <system response>
Complex	(combinations of the above patterns)

Note: *In this table from slide 26, the word "system" refers to the system being specified, which may be a subsystem or component of a larger system.*

Here are some examples of the various requirement types listed, written using the corresponding syntax pattern.

Ubiquitous

The FCC shall control communication on the Avionics Bus in accordance with MIL-STD-1553B and Table 3.1 of the program ICD.

Event-Driven

When the power button is depressed while the system is off, the system shall initiate its start-up sequence.

Unwanted Behaviour

If the battery charge level falls below 20% remaining, **then** the system shall go into Power

Saver mode.

State-Driven

While in the Power Saver mode, the system shall limit screen brightness to a maximum of 60%.

Optional Feature

Where the car is furnished with the GPS navigation system, the car shall enable the driver to mute the navigation instructions via the steering wheel controls.

A word about ubiquitous requirements

Many requirements that may seem ubiquitous are really driven by some trigger or condition. For example, the requirement:

The system shall monitor the engine temperature sensor and illuminate the engine overtemp symbol within 0.2 seconds of an overtemp indication.
is written in the ubiquitous format, but is, in fact, driven by an unwanted behaviour. Rewriting the requirement in the unwanted behaviour format makes the trigger-response nature of the requirement more clear:

If the engine temperature sensor indicates an overtemp condition, then the system shall illuminate the engine overtemp symbol within 0.2 seconds.
Be sure to check all “ubiquitous” requirements – especially if they’re functional requirements – for hidden triggers. Most true ubiquitous requirements are non-functional.

12. Go Beyond Expected Events and Behaviour

During a test flight over the Mojave Desert on Oct. 31, 2014, an unanticipated cockpit switch action by the co-pilot prompted the air brakes of Virgin Galactic's VSS Enterprise experimental spacecraft to deploy at 1.4 times the speed of sound. This unfortunate and *preventable* event resulted in the catastrophic, in-flight breakup of the vehicle, the death of the co-pilot and severe injury to the pilot.

Mistakes and oversights happen, but they can be greatly reduced by going beyond expected behaviour and anticipating exception scenarios. Exception scenarios are conditions in which a given requirement should not apply or should be altered in some way.

In Virgin Galactic's case, having an exception scenario for at least each phase of flight with corresponding triggers could have eliminated the system flaw that caused the airbrake to deploy at the wrong moment.

An example of a trigger condition and a corresponding trigger could be:

Trigger Condition: Spacecraft true airspeed between x and y.

Trigger: Air brakes shall not deploy.

If this were the only exception scenario identified, the requirement for deployment of the airbrake might have been corrected with the simple inclusion of the phrase:

"…except when the spacecraft true airspeed is between x and y."

On the other hand, if multiple exception scenarios were identified, it might be better to create a bulleted list of exceptions, in order to make the requirement easier to read.

13. Don’t Use Weak Words

Weak words – also called subjective, vague or ambiguous words – are adjectives, adverbs and verbs that don’t have a concrete or quantitative meaning. Such words are thus **subject to interpretation** by the reader of your requirements document.

Examine the following requirement:

Operation and location of all hands-on throttle controls shall be intuitive for both crew members.

What does “intuitive” mean in this case? It could mean something entirely different to the client or manager than it does to the design engineers. And what may be deemed “intuitive” by one user, could “require some getting used to” for another.

Good requirements are free of weak, subjective words such as:

- efficient
- powerful
- fast
- easy
- effective

- reliable
- compatible
- normal
- user-friendly
- few

- most
- quickly
- timely
- strengthen
- enhance

Define your requirements in precise, measureable terms. Don’t specify that a system or feature will *be* intuitive, reliable or compatible; define what will *make* it intuitive, reliable or compatible.

14. Avoid Passive Voice

Many adjectives that are also past participles of verbs – words like enhanced, strengthened and ruggedized – are notorious weak words, because they *sound* like engineering terms, but are weak in specificity. Here's an example:

The spacecraft shall be enhanced to protect the crew from an impact force of 400kg

What does enhanced mean in this case? Shall the spacecraft's fuselage be reinforced? Shall it have abort functionality? Shall it perform some manoeuvre to protect the crew? The word "enhanced" is ambiguous.

The problem here, however, is not so much the use of a weak word as it is the use of *passive voice* (indicated by a form of the verb "to be"). The phrase "shall *be* enhanced" seems to imply that this is a functional requirement, something that needs to be done. But in fact, it is not something that needs to be done *by* the system, but *to* the system. Thus it is not a *functional* requirement of the system, but a *quality* requirement – a *constraint* placed upon the implementation of the system.

This requirement could have been made more easily recognizable as a constraint if it had been re-phrased using the word "must" as follows:

The spacecraft must protect the crew from an impact force of 400kg.

– OR –

The spacecraft cabin must withstand an impact force of 400kg in order to protect the crew from injury.

Of course, the addition of a rationale statement (see Tip #8) would help to clarify this requirement further, but as you can see, just changing from *shall+passive* to *must+active* makes it clear that this requirement is a constraint and also makes it more implementation-neutral (see Tip #7).

15. Use Negative Requirements Sparingly.

While it is sometimes appropriate to state what a system shall not do, bear in mind that a system *shall not* do far more than what it *shall* do.

Stating requirements using “shall not” often causes reviewers to call into question other things the system shall not do, since “shall not” turns inaction or a lack of response into a requirement. Such confusion can generally be avoided by heeding the following rules of thumb.

- Use negative specification primarily for *emphasis*, in prohibition of potentially hazardous actions. Then state the safety case in the rationale for the requirement.
- Don’t use negative specification for requirements that can be restated in the positive. Substitute *shall enable* for *shall not prohibit*, *shall prohibit* in place of *shall not allow*, and so on.
- Avoid double negatives completely. Use *shall allow* instead of *shall not prevent*, for example.

16. Define Compatibility

Requirements documents often don’t give compatibility issues the emphasis they deserve. It is common to find requirements such as:

The in-vehicle infotainment system shall be compatible with the following devices…

But what, exactly, does “compatible” mean in this case? Does it mean the infotainment system shall be able to play music stored on connected devices? Shall it allow the driver to make hands-free phone calls from such devices? Is the vehicle required to have both wireless and wired connections?

If the system being designed must be compatible with other systems or components, explicitly state the specific compatibility requirements.

In other words, don’t leave it up to the hardware and software engineers to determine what will make the system they’re designing “compatible” with a given device (and expect the test engineers to make the same determination). It’s up to you, the requirements engineer, to define what it means to be compatible with the device in question.

17. Avoid Using Slash (/) Symbols

What does a “/” really mean? Does it mean and, or, one of, or a combination thereof (and/or)? These symbols can make all the difference between a clearly defined requirement and one that is impossible to interpret. In general, it is best to avoid using slash (/) symbols in stating requirements.

An example of ambiguity arising from the use of “/” is:

The vehicle shall enable the driver to manually disengage the automatic cruise/steering system with one hand via controls on the steering wheel.

In this example, it is unclear if the design engineers should provide for the cruise control and the automatic steering assist to be disengaged at the same time with a single one-handed action, or separately, via two one-handed actions. Probably, it’s the latter, in which case you really have two requirements which should be state separately:

X.X.X.1: The vehicle shall enable the driver to manually disengage the automatic cruise control function with one hand via controls on the steering wheel.

X.X.X.2: The vehicle shall enable the driver to manually disengage the automatic steering assist function with one hand via controls on the steering wheel.

Slash symbols should act as red flags, signalling the need to watch out for ambiguities. If, as in the preceding example, a subsystem is named with a slash because it’s multifunctional, ask yourself if referring to its discrete functions or components – rather than the subsystem by name – might make your requirement more clear.

18. Don’t Fall into the Requirements Document Vagueness Trap

Requirements *specify* the expected behaviour and essential properties of a system. So, given that the verb specify, the noun specification and the adjective *specific* all share the same root, it stands to reason that requirements should be specific, rather than vague. Does it not?

Yet, vagueness is epidemic in requirements specifications.

One of the big reasons for this is that both authors and customers often *allow* vagueness to slip into their requirements. Customers may like a vague requirement, reasoning that if its scope is unbounded, they can refine it later when they have a better idea of what they want. Authors and engineers may not mind, since a slack requirement may appear to give them more “freedom” in their implementation.

All eventually suffer, however, when the implementation misses the mark and extensive rework is required.

Here are four simple pointers for avoiding vagueness:

- Use active voice (shall + present tense verb) and avoid passive voice (shall *be* + past participle) wherever possible (see Tip # 14).
- Do not use unspecific adjectives (weak words) such as easy, straightforward, or intuitive (see Tip #13).
- Define *precisely* what the system needs *to do* (in functional requirements) or *to be* (in non-functional requirements) in such terms that compliance can be readily observed, tested or otherwise verified (see Tip #6).
- Don’t be swayed by those who want to keep requirements vague. Keep in mind the costs of scrap and re-work while defining requirements.

19. Write Requirements Documents from the Perspective of a Client or Manager

Requirements are intended to be the *control system* that keeps your development aligned with your customer's or manager's expectations.

This might sound obvious, but many engineers are so focused on authoring requirements with a certain concept in mind, they forget to adequately consider the product from the perspective of the customer or manager who needs to make sure the system can be easily and cost-effectively used and maintained.

Such a perspective can't be narrow. It comes from a thorough analysis of the needs of all potential stakeholders who will interact with the system. The list of these stakeholders may well go beyond what had been initially considered and should take into consideration all relevant domain experts, and even users!

For an avionics component, for example, you and the rest of your requirements development team would want to ask yourselves questions like:

- Which other components will this component interface with?
- Will this component interface with third-party suppliers' systems?
- Which maintenance crews will come into contact with this?
- Do the pilots need to interact with it?

Identify your stakeholders early, consider their use levels, and write from their perspective.

20. Evaluate the Requirements Document with a Diverse Team

Besides writing requirements from the perspective of a client or manager, another requirements quality best practice is to evaluate requirements with a diverse team.

This team should consist of any designers and developers who will be using the requirements to create the system, the testers who will verify compliance with the requirements, engineers who design, maintain or manage other systems that will support or interact with the new system, end-user representatives and, of course, the client team.

Many companies require just such an evaluation – and a formal sign-off of the requirements document – by all affected internal organizations, *before development can begin*. Any subsequent additions or changes to the document undergo a similar evaluation as part of a formal change management system. Such a system greatly increases the probability that the requirements will meet the needs of all stakeholders.

Tip 20a: Make note of which users were heavily considered for each requirement, so you can have that user provide focused feedback only on the requirements that are relevant to them.

21. Don’t Hand Off the Requirements Document for Verification Before Completing a Quality Check

Most professionals wouldn’t dream of handing in a report without proofing it for spelling and grammar errors. Yet, many requirements documents make it to the verification stage without undergoing any prior quality checks for completeness, consistency and clarity.

Having a quality assurance checklist to use in rechecking your requirements document greatly streamlines the process of making sure it conforms with best practices. That’s why we’ve created just such a checklist – based on the previous 20 tips in this guide – which is now available for download!

To ensure an exceptionally clear requirements document that is a dream to work with, be sure to check it against your checklist prior to submitting it to your verification team.

101-6080 Young Street, Halifax, Nova Scotia, Canada B3K 5L2 tel: 902.422.0212