ALTEN TECHNOLOGY

# WRITING EFFECTIVE REQUIREMENTS

## Reducing Risk and Ensuring Positive Outcomes Through Clear and Concise Specifications

by Jared Dean and Kyle Harrison
Updated 13 March 2024

10875 N Dover St #500
Westminster, CO 80021

# THE PROBLEM

Many times, the success (or failure) of a product development effort is determined before any engineering or technical execution takes place. Ill-defined, under-defined, and unclear requirements can lead to misguided implementation, disagreement around application, and worst of all, delays in product releases. Especially when working with tight deadlines and tighter budgets, knowing how to write good requirements can ensure positive outcomes and common understanding of scope and complexity where there otherwise may be none.

# EXECUTIVE SUMMARY

As providers of engineering services, we understand the importance of having a well-defined vision for new product development and that vision's impact on the success of individual projects. Similarly, internal product development efforts (sans outsourcing) also need a good set of practices surrounding the development of product requirements. In this paper, we describe the ALTEN Technology method for writing good product requirements as well as how to capture, document, track, and manage them throughout the project life cycle. Using real project inputs and outputs, we explore both good and bad examples of product requirements.

## WHY CAPTURE REQUIREMENTS?

In every product development effort, requirements—both acknowledged and implied—drive tasks (and costs) that must be executed and addressed prior to finalizing and releasing the product. Simply discussing requirements at a high level or half-heartedly documenting an outlined view of general requirements is not only a costly, bad practice but may violate industry regulations. Particularly for work with an engineering services firm, requirements will not only define the project's scope but also serve as the initial understanding of what it will take to complete the project and how success and completion will be defined.

# CAPTURING REQUIREMENTS

## THE PRODUCT REQUIREMENTS SPECIFICATION (PRS)

A good requirements document is a succinct, complete summary of known design inputs. Each requirement included in the PRS adds additional cost to the project (regardless of internal or external execution). Because of the cost of additional specifications, the goal should be to create the fewest requirements possible while accurately capturing the end customer's needs. This can be particularly challenging as a service provider because we aim to exceed expectations, and the natural tendency for engineers is to gravitate toward new, cutting edge, and exciting technologies and implementations—but we must focus on the end goal.

The PRS comprises crucial information regarding the requirements and their relative state. The most common elements used to manage, track, and implement the requirements include the following:

**ID:** A numerical identification number for a requirement, used throughout the project for reference.

**Epic or Component:** The subsystem or feature that a requirement is part of (such as database, reporting, or user interface).

**Specification:** The requirement statement.

**State:** For less formal tracking, generally, either proposed or firm.

**Verification Activity:** The ID number of the verification activity that will test the requirement.

**Source:** The originating document from which the requirement was derived.

## INITIAL REQUIREMENTS

An obvious question is "where do I start?" A good place to begin mining requirements is to request existing documentation, such as preliminary requirements or specifications, request-for-quote, VOC and marketing documents, and regulatory design documents. In the absence of these documents, or as follow-on, it is best practice to discuss, illustrate, and capture typical use cases and functional flow block diagrams for the product in question. Simply walking through the intended use of the product—even simple tasks like turning the device on—can generate a multitude of requirements. Take, for example, a medical device intended to monitor cardiac rehabilitation patients in a clinical setting. The client states that the end user "turns the device on and begins monitoring the patients." Had we not discussed in detail what that means and how it is done, we never would have determined that the workstation is in a different room from the patients and the clinician must have wireless access to the patient database to process that day's activities.

## WHERE DO THEY GO?

Soon after the project kickoff, the project manager or systems engineer should determine the best vehicle for capturing, tracking, and managing requirements. For most regulated industries, requirements can be captured in a formal document that is stored in the project's design history file (DHF) and approved by predetermined parties prior to release. For other industries or projects not bound by strict regulations, a simple Excel file or Microsoft SharePoint site is adequate for managing requirements. We will discuss the vehicle for capturing, maintaining, and managing requirements in later sections.

## TYPES OF REQUIREMENTS

Not all requirements are created equal. In fact, there are different types of requirements to be aware of because they affect implementation, verification, and, ultimately, project cost. The following requirement types are typical among complex, technology-driven products:

### Functional Requirements

These requirements specify what the device does, focusing on the operational capabilities of the device, the processing of inputs, and the resulting outputs.

**Example:** The device shall output test results for duration, average force, and serial ID number in a .csx format.

**Performance Requirements**
These requirements are quantitative and specify how much or how well the device must perform, addressing issues such as speed, strength, response times, accuracy, and limits of operation. These requirements include a quantitative characterization of the use environment, including temperature, humidity, shock, vibration, and electromagnetic compatibility. Requirements concerning device reliability and safety also fit into this category.

**Example:** The device shall remain fully operational in temperatures ranging from 0 ˚C to 50 ˚C.

**Interface Requirements**
Interface requirements specify characteristics of the device that are critical to compatibility with external systems; specifically, those characteristics that are mandated by external systems and outside the control of the design team. One interface that is important in every case is the user or patient interface.

**Example:** The device shall be compatible with 6-lead ECG cable part number xxx-xxx.

# DEVELOPING REQUIREMENTS

## WRITING GOOD REQUIREMENTS

The PRS is of limited use if the individual statements are poorly worded. Writing good requirements necessitates concise, simple language and precise wording. The goal of a requirements statement is to be clear and unambiguous to anyone who reads it.

Most of the requirements should be written in complete, declarative sentences with an imperative voice and end with a period. A declarative sentence simply states a fact or idea without requiring an answer or action on the part of the reader. Declarative sentences should not question or elicit an emotional response. The imperative voice is commonly used in engineering writing, and it is especially important in specification writing. The imperative voice expresses a command and focuses attention on the verb more than the subject. Examples of good and bad specification writing follow:

**Bad 1:** If the user pushed a button, 2 cc of fluid “A” shall be added to the cup by the system.

**Bad 2:** In proper operation, the user will request and receive 2 cc of fluid “A” in the cup.

**Good:** The system shall fill the cup with 2 cc of fluid “A” upon receiving a user request.

Requirements should be broken down into single, testable statements. If a requirement cannot be verified with testing or some other type of verification activity (such as analysis or demonstration), then it is a meaningless requirement and may cause potential problems during verification testing. Avoid combining multiple requirements into one statement wherever possible. The body of the requirement statement should not document the source or rationale for a requirement. Some examples follow:

**Bad:** External hardware to be tamper-resistant. Hardware interior to units is not restricted to tamper- resistant variety.

**Good:** External hardware shall be #4 pinned-Torx head style. Interior hardware shall be #2 Phillips head style.

**Bad:** One component of the solution has a significant $TiO_2$ component. This component is known to cause corrosion and build-up on ball valves and filters.

**Good:** The system shall not use ball valves or filters.

Avoid defining implementations with the requirement. A requirement should define how the system functions, how well it performs its intended purpose, and the budgetary and physical constraints placed on the design team. A requirement should not force the design team to solve the design challenge with a specific technology or approach unless specifically requested by the client. Although some clients do request inclusion of particular parts or hardware, the specification writer must be careful not to create unintended constraints.

**Bad:** Front of base station will incorporate at least one touch sensor to verify that a car is present for charging.

**Good:** The base station shall verify that a car is present before charging.

**Bad:** Design shall have some sort of tamper-proof seal incorporated into the design. The intent is to allow the manufacturer to know whether the guts of the unit have been tampered with.

**Good:** The system must provide a method to visually indicate whether an individual has accessed the internal mechanism of the unit.

Verb tense is important. All requirements are not created equal. The following list is an excerpt from the INCOSE Systems Engineering Handbook[1] and describes the common auxiliary verbs and forms of the verb *to be* as they apply to specifications:

- **Shall** - *Shall* requirements are demands upon the designer and the resulting product.
- **Will** - Statements containing *will* identify a future happening. *Will* is used to convey an item of information explicitly not to be interpreted as a requirement.
- **Must** - *Must* is not a requirement but is considered a strong desire or possible goal of the client.
- **Other forms** - *To be*, *is to be*, *are to be*, *should*, and *should be* are indefinite forms of the verb and should be minimized when developing requirements. These are not requirements, but should be considered capabilities desired by the client.

Keep language simple and avoid playing with synonyms. Typically, good writing is characterized by using a variety of sentence constructs, words, and phrases. When writing a specification, use the simplest, most common words and phrases. In addition, once you have used a word, avoid using synonyms in subsequent specifications. Synonyms never have exactly the same meaning and can cause unintended confusion.

There are a few other common mistakes that should be avoided. Below is a list of some of the more common errors made when writing specifications:

- Avoid superlatives (e.g., *best* or *most*) and ambiguous qualifiers (e.g., *significant*, *minimal*, *approximately*, or *real-time*).
- Avoid comparative specifications (e.g., *better than* or *faster than*).
- Do not add loopholes to a specification (e.g., *if possible* or *as appropriate*).

[1] Haskins, C., ed. 2010. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities.* Version 3.2. Revised by M. Krueger, D. Walden, and R. D. Hamelin. San Diego, CA (US): INCOSE.

# MANAGING REQUIREMENTS

## EVOLVING OR CHANGING REQUIREMENTS

It is not uncommon that requirements need further discussion, disposition, or research prior to solidification. Requirements may evolve during the project, and changes must be properly managed and documented. This is particularly common in new product development efforts where the technology is also under development and not heavily based on an existing product. When possible, requirements should be firm prior to moving into design and development.

In the event a requirement evolves, needs to be changed, or, in some cases, a new requirement needs to be added, it is imperative that the team review the updated requirement and its impact on the project from technical, budgetary—including cost of goods sold (COGS)—and scheduling perspectives. Regardless of whether there is a direct cost impact, the change should be captured in a formal engineering change order to provide traceability and agreement that the requirement has changed or the project scope has been updated.

Formally tracking the changes makes it clear to the design team, project sponsor, and stakeholders that the project has been redefined, even if only in a minor way.

Following disposition of the updated requirement, the PRS should also be updated and released as the latest revision in addition to affected verification and traceability documentation.

## TRACEABILITY AND VERIFICATION

In addition to providing guidance to the design team, requirements also serve as the defining component for traceability and verification, especially in regulated environments.

Each requirement is mapped back to a specifically defined verification activity that will serve as evidence that the requirement has been satisfied. This is for the formal verification execution at the end of the project and also a convenient way for the design team to test changes *ad hoc* as they are made—substantiating a higher quality product. In some cases, the verification activity will be a component of the PRS. In others, it may be a stand-alone verification plan. Regardless, the requirement and verification activity references shall remain consistent for traceability. An example is provided below.

**Requirement 3.9.7: Crosshead Travel**
The crosshead travel of the device shall be maximized at 500 mm.

**Verification Matrix:**

| REQUIREMENT | TEST METHOD | TEST LEVEL | PROTOCOL |
|---|---|---|---|
| 3.9.7 Crosshead Travel | Test | System | 4.2.2 |

The above example shows the requirement statement as seen in the specification section of the PRS as well as the verification matrix from the verification section of the PRS. In this case, the team has indicated that a test procedure will be conducted at the system level to verify requirement 3.9.7 and that the procedure can be found in protocol 4.2.2.

This example provides auditable evidence of traceability. The management of the requirements tells a story to interested parties and answers questions about the origin and verification of the requirements.

## FREQUENTLY ASKED QUESTIONS

**What if I know a requirement is coming for a particular area but don't know what the actual requirement is yet?**

At times, the client may not be prepared to provide exact requirements for the PRS (pending VOC or needed input). The best practice is to add a placeholder item in the PRS document. Give the specification an appropriate title, and put TBD in the description field. By consistently using TBD in the description field, you allow yourself to easily search and sort later. Set the status to unknown.

**Is this requirement essential?**

Per the Code of Federal Regulations, Title 21, "those design outputs that are essential for the proper functioning of the device" shall be identified.[2] In general, any requirement that deals with human safety is essential. In practice, the client should decide whether the requirement is essential. Use of the essential category in the PRS is only required under the full control process. Check the project Model Control Checklist to determine whether the column is required.

**What do I do if I've been provided a poorly written specification?**

At times, clients have been hesitant to request that ALTEN Technology execute a requirements phase for a project, believing they have a good grasp of the requirements. They may even have them documented. However, requirements must be written properly to avoid costly confusion in the future. The best response is to modify or clarify them, then review them with the client to make sure that the updates have maintained the original intent. Do not change client-supplied requirements without client review and approval. Reserve the *firm* status only for those requirements that are unambiguous, quantitative, verifiable, and client approved.

# CONCLUSIONS

Through the methods and best practices discussed in this paper, we have shown that product developers and design teams can benefit from writing clear, concise, and effective requirements. Moreover, identifying, documenting, and establishing those requirements early in the project will ensure a common understanding of the scope and cost of the effort. Proper change management will minimize future disagreements and conflict. By writing quantifiable and declarative requirements and tracing them to verification activities, the team members will know when they have designed the right product and will have appropriate evidence to justify confidence in the product's quality upon its release into the field. Even for less formal or less-regulated product development efforts, writing effective requirements is a best practice, especially when dealing with outside contractors, vendors, and suppliers.

### UNAMBIGUOUS REQUIREMENTS LEAD TO HAPPY OUTCOMES

Although at times it may feel tedious, over-detailed, or even like a duplication of efforts, we have found that by routinely reviewing and capturing effective requirements, we can establish much better relationships with our clients and teams. Design teams appreciate a focused approach that provides freedom in creativity (a luxury not often experienced when working on projects with a fixed price or schedule), and clients appreciate the predictable results and smooth-running projects.

# ABOUT ALTEN TECHNOLOGY

ALTEN Technology is an engineering services company that provides innovative solutions for engineering, IT, and product development projects across the product life cycle. For decades, ALTEN Technology has been helping clients develop products that are changing the world. We provide support across industries, including aerospace, defense, automotive (including commercial vehicles), medtech, life science, rail, energy and environment, robotics, and unmanned systems.

**Want more information?** This white paper is an overview of the best practices identified by the International Council on Systems Engineering (INCOSE) as well as ALTEN Technology. For more information, see the Systems Engineering Handbook, Technical Process section, available at www.incose.org, or contact ALTEN Technology directly.

[2] 21 CFR § 820.30(d) 2024