8 Do's and Don'ts for Writing Requirements

Every word matters when authoring requirements. Something as simple as adding an adverb or using “should” instead of “must” can create ambiguity that confuses engineers and sets a project back.

Let’s see how you can write requirements that are both clear and traceable across the product development lifecycle.

DO: Use a Requirements Template

A template gives consistent structure to requirements. It can be in a user story or systems engineering format, either of which provides uniform construction to support easier testing.

DON’T: Use Adverbs

“Quickly,” “easily,” and other adverbs don’t provide clear guidance to testers. Instead, **focus on acceptance criteria that are testable and measurable.**

DO: Standardize your Language

The English language contains numerous words with similar meanings in everyday usage. Settle on a few to represent agreed-upon meanings, like “shall” for binding high-priority requirements.

DON’T: Be Ambiguous

Requirements are often ambiguous because they’re too general, e.g., “the device shall be easy to use.” **Get more specific,** whether that means setting a clear benchmark or naming a specific color.

DO: Use Active Voice and Specific Adjectives

Use active voice verbs. For instance, “the car shall withstand...” is clearer than “the car shall be enhanced to withstand...” Also select specific adjectives instead of standbys like “user-friendly” and “compatible.”

DON’T: Mix Design Specifications into Requirements

When possible, aim to remove design from requirements, as the latter describe a need while the former constitute a response to that need. **Design-free requirements give engineers more freedom.**

DO: Regularly Review Requirements with Stakeholders

Reviewing your requirements with others is a reliable way to ensure shared understanding. Collaborating within a real-time platform lets teams exchange feedback, ensure testability, and minimize rework.

DON’T: Rely on Negative Requirements Statements

Negative statements can introduce ambiguity, since there are virtually infinite things that any system will “not do” en route to fulfilling its positive requirements. **Check negative statements carefully and use them sparingly.**

www.jamasoftware.com