//Junaidsaeed

//Roll#153172

Ambiguous Requirements:

Many software requirements suffer from ambiguity. Ambiguity means that a single reader can interpret the requirement in more than one way or that multiple readers come to different interpretations. In either case, ambiguous requirements lead to confusion, wasted effort and rework.Ambiguous requirements lead to confusion, wasted effort and rework. An article from software requirements describes several common types of requirements ambiguity and suggests how to avoid them.

Example-1:

We command line

When writing requirements that are clear, always use the present with indicative mood. In other words, it happens now because I command you to do it. Good requirements use must, shall, and can. They do not use could, should or may

Example#2:

Negative requirements

Negative, or inverse, requirements state what the system will not do. Here's an example from an actual project: "All users with three or more accounts should not be migrated." Try to rephrase negative requirements into a positive sense: "The system shall migrate only users having fewer than three accounts." When changing a negative requirement into a positive one, you often need to insert the word "only" to clarify the conditions that permit the system action to take place. Double and triple negatives are especially confusing; avoid them in all situations.

Example#3:

Adverbs

Adverbs are subjective and qualitative, and they inevitably result in diverse individual interpretations. Here's an illustration from an actual specification: "Generally incurs a 'per unit' cost…" But this requirement did not provide any indication regarding the conditions under which we would not incur a per-unit cost or what to do then. I don't know how to determine whether we've satisfied a requirement that says "Provide a reasonably predictable end user experience." I guess it doesn't have to be completely predictable.