Avoid Ambiguous Requirements, See Unicorns

These are 7 steps to avoid the ambiguous requirement and maybe see a unicorn. "The perfect requirement is like a unicorn. You really wish it existed, but it doesn't. Get over it".

MAY 20, 2015

Christine Hart
Principal Marketing Technology Consultant

SHARE

Finding the unicorn - Image courtesy to ©yosuke muroya
https://www.flickr.com/photos/hamur0w0/6833346648/

A jaded colleague (not at Netcentric!) once told me this. He was right. Requirements constantly change.
One thing you can do is tape a horn on that pony of a requirement and call it a unicorn by eliminating ambiguous language and ensuring clarity. Even as the sands of client changes and demands are constantly shifting under your feet, making your requirements clear and precise helps your developers, your clients, your partners and your bottom line.
For example, I was working with a client and they told me that they needed to be able to upload "large files." I started investigating solutions into what I thought they thought was a "large file". We found a solution with a partner who assured the team that their platform could handle the "large files." I assured the client that we had a solution and we began testing. They were giving us feedback that the solution was not meeting their needs, was freezing and full of bugs. To the client "large" meant 20-50GB and to our team and the platform provider "large" meant up to 5GB.
It is easy to see in retrospect how this mismatch happened but this happens all the time in requirements! Ambiguous language is the plague of requirements and ruins specifications.
Here are # of hints and tricks to help you turn your requirements into unicorns, sort of.
What exactly is ambiguity?
According to the Merriam Webster definition ambiguous is:
a - doubtful or uncertain especially from obscurity or indistinctness <eyes of
an ambiguous color>
b - inexplicable
capable of being understood in two or more possible senses or
ways <an ambiguous smile> <an ambiguous term> <a deliberately ambiguous reply>
And its inverse, unambiguous:
: not ambiguous : clear, precise <unambiguous evidence>
The key here is the second definition. "Capable of being understood in two or more possible senses" like my example from a above. "Large" meant something very different to myself and to the client. To avoid any possible misinterpretation, we want our requirements to mean the exact same thing to readers of a similar background. Here at Netcentric we have to truly take that into account since we represent more than 29 nationalities within our staff and work for clients all across Europe. How do we achieve our unicorns?

1. We are not vague

Linguistic vagueness is a particular problem in software development. A requirement is vague if it is not clear how to measure if the requirement has been fulfilled or not. A non-functional require suffers from the fate of vagueness far more than functional requirement. For example, the non-functional requirement of fast response time is vague because there is not precise way of describing or measuring "fast response time", just like "large file". To avoid this, use as specific of measurements as possible. For example "fast response time" is better represented by writing "response time at or below 500ms." When a project has a majority of requirements that are vague, we wonder if the true requirements have been captured at all.

We also try not to use acronyms in our requirements. Not every one knows that SDLC means what it means. Every client has their own sets of acronyms and using them will cause problems for the next team to work on the project and makes the on-boarding time take much longer. Not only does the new Project Manager have to read the all the documentation, but they have to decipher what the heck CID or CRM or RAWS are.

2. 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.

3. Pronouns are a no-no

We use the subject at all times even when it is repetitive. We are not writing poetry. We also avoid using more than one conjunction in a sentence and we don´t use conjunctions with a modifier. Not sure what a modifier is? See below:

I sat with the dog and the cat and the bird sat with me.
Here, we are not sure if I sat with the only the dog or if I sat with the dog and the cat and the bird or if I sat with the dog and the cat.

The black dog and cat were walking.

In this example the modifier is "black" and the placement makes the sentence unclear. Are both the dog and cat black? We can assume, and most do that this means just the dog, but not all readers would assume that.

4. We are precise

Avoid using general descriptors for characteristics or amounts. We don´t like to use various, some, any, a lot, every or up to. We also avoid using vague words for behaviours like possibly, probably, and usually. We also like to use the industry standard amount for things when we can or the exact pixel amounts for front-end elements.

Trigger an automatic log out when a user attempts to download up to 5 times.
Display is up to large screen size.

Is this 5 times including the 5th time or 4 times? What if the download is part of the design of this part of the program? For example the page has a document that is in 5 parts. For the screen size, 720px may be large for some and 1280 may be large for some. It is better to get as specific as possible which leads me to the next point.
In my example, the requirement around "large files" should have read:

The platform will handle upload and download files up to and including 50GB in size.

That requirement is starting to look more like a pony with a nice paper horn taped to its head!

5. Get the context

Don't depend on external context. The people reading the requirements may not have been in the same meeting that you were and the developers don't generally have the kinds of access to the clients that the writers of requirements do. A missing context can create a vital flaw in the requirements. Is that large screen specification for a region in China that the majority of screens are still only up to 720px? That would make all the difference for this requirement.

6. How won't it work

We use inverse requirements to describe the functionality that the product will not perform. Inverse requirements are often misused to express non-functional requirements, e.g., the system must not lose user data, which is actually a reliability requirement. However, in its essence, an inverse requirement rules out possible interpretations of one or more functional requirements. Th inverse requirement disambiguates the functional requirements. A true inverse requirement has no test case. If you can get a test case for an inverse requirement, the inverse requirement is most likely a non-functional requirement.

Requirement: The vending machine offers refreshing drinks.
Inverse requirement: The vending machine does not offer tea, coffee, and alcoholic drinks.

7. Validity

We know that whether something is valid or invalid is not up to us. We have to include firm criteria for validation. For example, we don´t say:

Upload will fail if the file is invalid.

In this case, what is invalid? Is it part of a set of unsupported files types? Then it is not invalid, just not part of the requirement.

Following these seven hints, will help you clean up your language and ensure that the requirements are crystal clear through the whole development chain. Your developers and testers will love you. Your clients will applaud you. And you might even see a real unicorn, eventually!