**General requirements for testing**

Requirements for software are usually grouped into a bewildering array of categories. Functional and nonfunctional requirements are on top, and a huge number of subcategories are underneath. These are three ways I considered before going to the main different tests requested, because I thought this is going to be useful to us as we go into the testing part. These requirements are to guide us through the software we will be using and it applies to any software. These includes: explicit, implicit, and latent requirements.

**Explicit Requirements**

“**The Things You Wrote Down”**Any time you’re comparing software to any kind of written documentation, you’re testing using explicit requirements. Remember, though, that an explicit requirement usually implies one or more negative checks that should also be performed.

When the software fails to match an explicit requirement, first examine whether it’s the software or the documentation that needs to change. It could be one or the other, or even both. If the software is reaching the testing stage without matching its explicit requirements, it’s worth taking a step back and examining your team’s process, too. Verification of explicit requirements should be handled by the developer writing the code, ideally by creating a set of automated checks demonstrating that those requirements have been satisfied.

**Implicit Requirements**

**“The Things Your Customers Will Expect”**Testing for implicit requirements is a lot trickier, both on the bug discovery side and the bug reporting side. It also represents a tester’s best opportunity to help the development effort.

To test for implicit requirements, a tester must become an expert in the customer’s problem domain and in the technology the software uses to solve those problems. It’s also difficult to demonstrate “coverage” when testing for implicit requirements.

When the software fails to match an implicit requirement, a report of that failure must also include an explanation of why a customer would expect the software to behave differently. What is the bug’s impact in terms of its effect on a customer’s experience?

Find out how the customers are using the software, and use that information to design scenario tests to discover latent requirements. Remember, too, that end users don’t always know what’s possible and might not ask for everything that can make them happy.

**Latent Requirements**

**“Things That Will Delight Your Customers”**Testing for latent requirements is the trickiest of all because it’s impossible to guess what those requirements will be until you get your hands on the software. To test for latent requirements, testers must deeply understand the customer’s preferences, while keeping in mind that they are not customers.

When the software fails to match a latent requirement, that failure represents an opportunity to improve the software. It also represents unplanned work. Rather than treating these opportunities as bugs, find a way to work them into your team’s regular planning process. A team with a fluid mechanism for incorporating new latent requirements will produce a more satisfying product and happier customers.