From Software Development

May 1999

Writing Good Requirements

All too often, software requirements are badly written and hard to follow. Clarifying your specifications will benefit everyone involved.

by Karl Wiegers

It looks like your project is off to a good start. Your team had customers involved in the requirements elicitation stage and you actually wrote a software requirements specification. The specification was big, but the customers signed off on it, so it must have been O.K.

Now you're designing one of the features and you've found some problems with the requirements: You can interpret requirement 15 a couple different ways. Requirement 9 states precisely the opposite of requirement 21; which should you believe? Requirement 24 is so vague that you don't have a clue what it means. You just had an hour-long discussion with two other developers about requirement 30 because all three of you thought it meant something different. And the only customer who can clarify these points won't return your calls. You're forced to guess what many of the requirements mean, and you can count on doing a lot of rework if you guess wrong.

Many software requirements specifications (SRS) are filled with badly written requirements. Because the quality of any product depends on the quality of its raw materials, poor requirements cannot lead to excellent software. Sadly, few software developers have been educated about how to elicit, analyze, document, and verify requirement quality. There aren't many examples of good requirements to learn from, partly because few projects have good ones to share, and partly because few companies are willing to place their product specifications in the public domain.

This article describes several characteristics of high-quality software requirement statements and specifications. I will examine some less-than-perfect requirements from these perspectives and take a stab at rewriting them. I've also included some general tips on how to write good requirements. Evaluate your project's requirements against these quality criteria. It may be too late to revise them, but you might learn how to help your team write better requirements next time.

Don't expect to create a SRS in which every requirement exhibits every desired characteristic. No matter how much you scrub, analyze, review, and refine requirements, they will never be perfect. However, if you keep these characteristics in mind, you will produce better requirements documents and you will build better products.

Characteristics of Quality Requirement Statements

How can you distinguish good software requirements from problematic ones? Individual requirement statements should exhibit six characteristics. A formal inspection of the SRS by project stakeholders with different perspectives is one way to determine whether or not each requirement has these desired attributes. Another powerful quality technique is writing test cases against the requirements before you cut a single line of code. Test cases crystallize your vision of the product's behavior as specified in the requirements and can reveal omissions and ambiguities.

Characteristic #1: They must be correct. Each requirement must accurately describe the functionality to be delivered. The reference for correctness is the source of the requirement, such as an actual customer or a higher-level system requirements specification. A software requirement that conflicts with a corresponding system requirement is not correct (of course, the system specification could be incorrect, too).

Only users can determine whether the user requirements are correct, which is why it's essential to include actual users, or surrogate users, in requirements inspections. Requirements inspections that do not involve users can lead to developers saying, "That doesn't make sense. This is probably what they meant." This is also known as "guessing."

Characteristic #2: They must be feasible. You must be able to implement each requirement within the known capabilities and limitations of the system and its environment. To avoid infeasible requirements, have a developer work with the requirements analysts or marketing personnel throughout the elicitation process. This developer can provide a reality check on what can and cannot be done technically, and what can be done only at excessive cost or with other trade-offs.

Characteristic #3: They must be necessary for the project. Each requirement should document something the customers need or something you need to conform to an external requirement, an external interface, or a standard. You can think of "necessary" as meaning each requirement originated from a source you know has the authority to specify requirements. Trace each requirement back to its origin, such as a use case, system requirement, regulation, or some other voice-of-the-customer input. If you cannot identify the origin, perhaps the requirement is an example of gold-plating and isn't really necessary.

Characteristic #4: They must be prioritized. Assign an implementation priority to each requirement, feature, or use case to indicate how essential it is to a particular product release. Customers or their surrogates have the lion's share of the responsibility for establishing priorities. If all the unprioritized requirements are equally important, so is the project manager's ability to react to new requirements added during development, budget cuts, schedule overruns, or a team member's departure. You can determine priority by considering the requirement's value to the customer, the relative implementation cost, and the relative technical risk of implementing it.

Many groups use three levels of priority. High priority means you must incorporate the requirement in the next product release. Medium priority means the requirement is necessary but you can defer it to a later release if necessary. Low priority means it would be nice to have, but you might have to drop it because of insufficient time or resources.

Characteristic #5: They must be unambiguous. The reader of a requirement statement should draw only one interpretation of it. Also, multiple readers of a requirement should arrive at the same interpretation. Natural language is highly prone to ambiguity. Avoid subjective terms like user-friendly, easy, simple, rapid, efficient, several, state-of-the-art, improved, maximize, and minimize. Write each requirement in succinct, simple, straightforward language of the user domain, not in technical jargon. You can reveal ambiguity through formal requirements specifications inspections, writing test cases from requirements, and creating user scenarios that illustrate the expected behavior of a specific portion of the product.

Characteristic #6: They must be verifiable. See whether you can devise tests or use other verification approaches, such as inspection or demonstration, to verify that the product properly implements each requirement. If you can't verify a requirement, determining whether or not it was correctly implemented is a matter of opinion. Requirements that aren't consistent, feasible, or unambiguous are also not verifiable. Any requirement that the product shall "support" something isn't verifiable.

Characteristics of Quality Requirements Specifications

A complete SRS is more than a long list of functional requirements. It also includes external interface descriptions and nonfunctional requirements, such as quality attributes and performance expectations. Look for the following characteristics of a high-quality SRS.

Characteristic #1: It is complete. The SRS shouldn't be missing any requirements or necessary information. Individual requirements should also be complete. It is hard to spot missing requirements, because they aren't there. Organize your SRS's requirements hierarchically to help reviewers understand the structure of the described functionality. This makes it easier to tell if something is missing.

If you focus on user tasks rather than on system functions when gathering requirements, you're less likely to overlook requirements or include requirements that aren't really necessary. The use case method works well for this purpose. Graphical analysis models that represent different views of the requirements will also reveal incompleteness.

If you know you are lacking certain information, use "TBD" (to be determined) as a standard flag to highlight these gaps. Resolve all TBDs before you construct the product.

Characteristic #2: It is consistent. Consistent requirements do not conflict with other software requirements or with higher level (system or business) requirements. You must resolve disagreements before you can proceed with development. You may not know which (if any) is correct until you do some research it. Be careful when modifying the requirements; inconsistencies can slip in undetected if you review only the specific change and not related requirements.

Characteristic #3: It is modifiable. You must be able to revise the SRS when necessary and maintain a history of changes made to each requirement. You must give each requirement a unique label and express it separately from other requirements, so you can refer to it clearly. You can make a SRS more modifiable by grouping related requirements, and by creating a table of contents, index, and cross-reference listing.

Characteristic #4: It is traceable. You should be able to link each software requirement to its source, which could be a higher-level system requirement, a use case, or a voice-of-the-customer statement. Also, link each software requirement to the design elements, source code, and test cases that implement and verify it. Traceable requirements are uniquely labeled and written in a structured, fine-grained way. Bullet lists are not traceable.

The difficulty developers will have understanding poorly written requirements is a strong argument for having both developers and customers review requirements documents before they are approved. Detailed inspection of large requirements documents is not fun, but it's worth every minute you spend. It's much cheaper to fix the defects at that stage than later in the development process or when the customer calls.

If you observe the guidelines for writing quality requirements (see sidebar) and if you review the requirements formally and informally, early and often, your requirements will provide a better foundation for product construction, system testing, and ultimate customer satisfaction. And remember that without high-quality requirements, software is like a box of chocolates: you never know what you're going to get.

Guidelines for Writing Quality Requirements

There is no simple formula for writing excellent requirements. It is largely a matter of experience and learning from past requirements problems. Here are a few guidelines to keep in mind as you document software requirements.

• Keep sentences and paragraphs short. Use the active voice. Use proper grammar, spelling, and punctuation. Use terms consistently and define them in a glossary.

• To see if a requirement statement is sufficiently well-defined, read it from the developer's

perspective. Mentally add the phrase, “call me when you’re done” to the end of the requirement and see if that makes you nervous. In other words, would you need additional clarification from the SRS author to understand the requirement well enough to design and implement it? If so, elaborate on that requirement before you proceed.

• Requirement authors often struggle to find the right level of granularity. Avoid long narrative paragraphs that contain multiple requirements. Write individually testable requirements. If you can think of a small number of related tests to verify a requirement’s implementation, it’s probably written at the right level of detail. If you envision many different kinds of tests, perhaps several requirements have been lumped together and should be separated.

• Watch out for multiple requirements that have been aggregated into a single statement. Conjunctions like “and” and “or” in a requirement suggest that several requirements have been combined. Never use “and/or” or “etc.” in a requirement statement.

• Write requirements at a consistent level of detail throughout the document. I have seen requirements specifications that varied widely in their scope. For example, “A valid color code shall be R for red” and “A valid color code shall be G for green” might be split into separate requirements, while “The product shall respond to editing directives entered by voice” describes an entire subsystem, not a single functional requirement.

• Avoid stating requirements redundantly in the SRS. While including the same requirement in multiple places may make the document easier to read, it also makes it more difficult to maintain. You must update multiple instances of the requirement at the same time to avoid inconsistency.

—Karl Wiegers