Why should I define requirements?

To deliver a successful software product, you need to develop, document, and validate software requirements. Properly understood requirements allow you to "begin with the end in mind" and are the basis for determining the success of your implemented software. After all, the purpose of software development is to satisfy users' needs, and these needs are precisely what the requirements define.

The price is high for not defining requirements or not doing it well. Poorly defined requirements result in *requirements defects*—errors in requirements caused by incorrect, incomplete, missing, or conflicting requirements. Defective requirements may result in:

- Cost overruns,
- Expensive rework,
- Poor quality,
- Late delivery,
- Dissatisfied *customers*, and
- Exhausted and demoralized team members.

Correcting defective requirements accounts for almost one-half of the cost of software development and is the most expensive kind of development error to fix. Defective requirements become multiplied in number and seriousness as they spread among multiple complex components in design, *code*, and tests. The result is a need for extensive and expensive rework, which costs from ten to one hundred times more to fix later in development.

To reduce the high *risk* of software project failure and the large costs associated with defective requirements, you must properly define requirements early in the software development process.

Requirements verification and validation

Requirements are critical to the success of the end product. Before you write the software's code, the emphasis is on the problem (i.e., defining what to build and ensuring that it is necessary to meet user needs). Although software tests are not executed during *requirements development*, performing conceptual tests will help to uncover incomplete, incorrect, and unclear requirements.

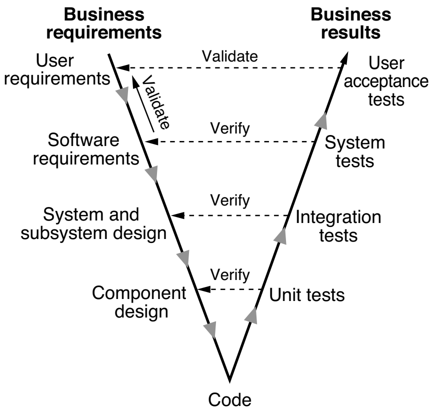
After you have begun to write the code, the emphasis is on testing the software solution against the requirements. Performing *user acceptance tests* will link the original needs back to business customers and end users, ensuring that the right product was built.

As requirements are developed, they are *verified* to see if they satisfy the conditions or specifications of the requirements development activity. *Verification* is like good management—it ensures that you *built the software correctly*.

When requirements are identified and later tested in user acceptance testing, they are *validated* to ensure that they meet user's needs. *Validation* is like good leadership—it ensures that you *built the correct software*.

Whereas *requirements verification* represents the development team's point of view—ensuring the software satisfies the specified requirements, *requirements validation* is concerned with the customer's point of view—ensuring the customer's needs are met.

How Requirements are Verified and Validated



What types of requirements are there?

Software requirements are broadly divided into functional and nonfunctional requirements. *Functional requirements* describe product capabilities—things that the product must do for its users or allow its users to do with the software. Functional requirements are the *doing* part of software—the actions, tasks, and behaviors that users generally interact with. They can be stated as:

- "The system shall provide the capability for schedulers to assign contractors to jobs in their local area."
- "The system shall permit the inventory manager to search for available inventory items."
- "The system shall notify the operator when the temperature exceeds the maximum set value."
- "The system shall store a log of temperature readings every three seconds."

Nonfunctional requirements are properties that the product must have that may not be evident to the user, including quality attributes, constraints, and external interfaces:

- *Quality attributes* describe properties of the software's development and operational environment, such as its performance, capacity, maintainability, portability, reliability, and usability. (See section 5.2 for more information on quality attributes.)
- *Design and implementation constraints* limit how the software can be designed. For example, a limit on the maximum number of concurrent users, the environment that the software will operate in, or a predetermined programming language to be used will all constrain the software design and implementation.
- *External interfaces* are the interfaces with other systems (hardware, software, and human) that the proposed system will interact with.

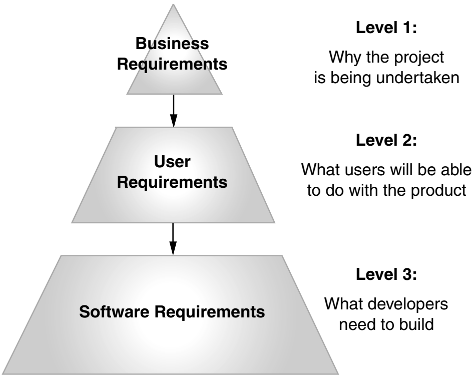
Nonfunctional requirements are the *being* part of the software—the characteristics and constraints for the software's behavior. They should be documented in quantifiable terms, such as:

- "The response time for loading all estimate information onto the screen shall be no more than six seconds after the user submits the estimate request."
- "During the peak holiday season between November 1st and January 5th, the inventory search capability shall permit 500 simultaneous users to search for inventory items."
- "The system's scheduling capability shall be available weekdays from 7 a.m. PST to 7 p.m. PST."
- "The system shall function on the following operating systems: Isis version 6 or higher and Grok version 2.0 and higher."

Where do requirements come from?

Software requirements operate on three levels: the requirements related to your business, those related to your users, and those that describe the software itself.

Requirements Levels


Business Requirements
Level 1:
Why the project is being undertaken
User Requirements
Level 2:
What users will be able to do with the product
Software Requirements
Level 3:
What developers need to build

Level 1: Business Requirements

Business requirements are statements of the business rationale for authorizing the project. They include a vision for the software product that is driven by business goals, business objectives, and strategy. Business requirements describe the high-level purpose and needs that the product will satisfy to increase revenue, reduce operating expenses, improve customer service, or meet regulatory obligations. The vision for the product provides a long-term view of what the end product will accomplish for its users and should include a statement of scope to clarify which capabilities the product will and will not provide.