Requirement

From Wikipedia, the free encyclopedia

In engineering, a **requirement** is a singular documented physical and functional need that a particular product or service must be or perform. It is most commonly used in a formal sense in systems engineering, software engineering, or enterprise engineering. It is a statement that identifies a necessary attribute, capability, characteristic, or quality of a system for it to have value and utility to a user.

In the classical engineering approach, sets of requirements are used as inputs into the design stages of product development. Requirements are also an important input into the verification process, since tests should trace back to specific requirements. Requirements show what elements and functions are necessary for the particular project. This is reflected in the waterfall model of the software life-cycle. However, when iterative methods of software development or agile methods are used, the system requirements are incrementally developed in parallel with design and implementation.

Requirements engineering is the set of activities that lead to the derivation of the system or software requirements. Requirements engineering may involve a feasibility study, or a conceptual analysis phase of the project and requirements elicitation (gathering, understanding, reviewing, and articulating the needs of the stakeholders) and requirements analysis,[1] analysis (checking for consistency and completeness), specification (documenting the requirements) and validation (making sure the specified requirements are correct).[2][3]

Contents

- 1 Product versus process requirements
- 2 Requirements in systems and software engineering
- 3 Product requirements
  - 3.1 Types of product requirements
  - 3.2 Characteristics of good requirements
    - 3.2.1 Verification
    - 3.2.2 Requirements analysis or requirements engineering
  - 3.3 Documenting requirements
  - 3.4 Changes in requirements
- 4 Disputes regarding the necessity of rigour in software requirements
- 5 See also
- 6 References
- 7 External links

Product versus process requirements

Projects are subject to three sorts of requirements:

- **Business requirements** describe in business terms *what* must be delivered or accomplished to provide value.
- **Product requirements** describe properties of a system or product (which could be one of several ways to accomplish a set of business requirements.)
- **Process requirements** describe activities performed by the developing organization. For instance, process requirements could specify specific methodologies that must be followed, and constraints that the organization must obey.

Product and process requirements are closely linked. Process requirements often specify the activities that will be performed to satisfy a product requirement. For example, a maximum development cost requirement (a process requirement) may be imposed to help achieve a maximum sales price requirement (a product requirement); a requirement that the product be maintainable (a Product requirement) often is addressed by imposing requirements to follow particular development styles (e.g., object-oriented programming), style-guides, or a review/inspection process (process requirements).

Requirements in systems and software engineering

In systems engineering, a **requirement** can be a description of *what* a system must do, referred to as a *Functional Requirement*. This type of requirement specifies something that the delivered system must be able to do. Another type of requirement specifies something about the system itself, and how well it performs its functions. Such requirements are often called *Non-functional requirements*, or 'performance requirements' or 'quality of service requirements.' Examples of such requirements include usability, availability, reliability, supportability, testability and maintainability.

A collection of requirements define the characteristics or features of the desired system. A 'good' list of requirements as far as possible avoids saying *how* the system should implement the requirements, leaving such decisions to the system designer. Specifying how the system should be implemented is called "implementation bias" or "solution engineering". However, *implementation constraints* on the solution may validly be expressed by the future owner, for example for required interfaces to external systems; for interoperability with other systems; and for commonality (e.g. of user interfaces) with other owned products.

In software engineering, the same meanings of requirements apply, except that the focus of interest is the software itself.

Product requirements

Types of product requirements

Requirements are typically placed into these categories:

- Architectural requirements describe what must be done by identifying the necessary system architecture of a system,

- Functional requirements describe the functionality that the system is to execute; for example, formatting some text or modulating a signal. They are sometimes known as capabilities.
- Non-functional requirements describe characteristics of the system that the user cannot affect or (immediately) perceive. Nonfunctional requirements are sometimes known as quality requirements or ilities.
- Constraint requirements impose limits upon the design alternatives or project/process operations. No matter how the problem is solved the constraint requirements must be adhered to.

Non-functional requirements can be further classified according to whether they are usability requirements, look and feel requirements, humanity requirements, performance requirements, maintainability requirements, operational requirements, safety requirements, reliability requirements, or one of many other types of requirements.

In software engineering this categorization is useful because only functional requirements can be directly implemented in software. The non-functional requirements are controlled by other aspects of the system. For example, in a computer system reliability is related to hardware failure rates, and performance is controlled by CPU and memory. Non-functional requirements can in some cases be decomposed into functional requirements for software. For example, a system level non-functional safety requirement can be decomposed into one or more functional requirements. See FURPS. In addition, a non-functional requirement may be converted into a process requirement when the requirement is not easily measurable. For example, a system level maintainability requirement may be decomposed into restrictions on software constructs or limits on lines or code.

Characteristics of good requirements

The characteristics of good requirements are variously stated by different writers, with each writer generally emphasizing the characteristics most appropriate to their general discussion or the specific technology domain being addressed. However, the following characteristics are generally acknowledged.[4]

Characteristic	Explanation
Unitary (Cohesive)	The requirement addresses one and only one thing.
Complete	The requirement is fully stated in one place with no missing information.
Consistent	The requirement does not contradict any other requirement and is fully consistent with all authoritative external documentation.
Non-Conjugated (Atomic)	The requirement is *atomic*, i.e., it does not contain conjunctions. E.g., "The postal code field must validate American *and* Canadian postal codes" should be written as two separate requirements: (1)

	"The postal code field must validate American postal codes" and (2) "The postal code field must validate Canadian postal codes".
Traceable	The requirement meets all or part of a business need as stated by stakeholders and authoritatively documented.
Current	The requirement has not been made obsolete by the passage of time.
Feasible	The requirement can be implemented within the constraints of the project.
Unambiguous	The requirement is concisely stated without recourse to technical jargon, acronyms (unless defined elsewhere in the Requirements document), or other esoteric verbiage. It expresses objective facts, not subjective opinions. It is subject to one and only one interpretation. Vague subjects, adjectives, prepositions, verbs and subjective phrases are avoided. Negative statements and compound statements are prohibited.
Mandatory	The requirement represents a stakeholder-defined characteristic the absence of which will result in a deficiency that cannot be ameliorated. An optional requirement is a contradiction in terms.
Verifiable	The implementation of the requirement can be determined through one of four possible methods: inspection, demonstration, test or analysis.

To the above some add Externally Observable, that is, the requirement specifies a characteristic of the product that is externally observable or experienced by the user. Such advocates argue that requirements that specify internal architecture, design, implementation, or testing decisions are probably constraints, and should be clearly articulated in the Constraints section of the Requirements document. The contrasting view is that this perspective fails on two points. First, the perspective does not recognize that the user experience may be supported by requirements not perceivable by the user. For example, a requirement to present geocoded information to the user may be supported by a requirement for an interface with an external third party business partner. The interface will be imperceptible to the user, though the presentation of information obtained through the interface certainly would not. Second, a constraint limits design alternatives, whereas a requirement specifies design characteristics. To continue the example, a requirement selecting a web service interface is different from a constraint limiting design alternatives to methods compatible with a Single Sign-On architecture.

Verification

All requirements should be verifiable. The most common method is by test. If this is not the case, another verification method should be used instead (e.g. analysis, demonstration or inspection or review of design).

Certain requirements, by their very structure, are not verifiable. These include requirements that say the system must *never* or *always* exhibit a particular property. Proper testing of these requirements would require an infinite testing cycle. Such requirements must be rewritten to be verifiable. As stated above all requirements must be verifiable.

Non-functional requirements, which are unverifiable at the software level, must still be kept as a documentation of customer intent. However, they may be traced to process requirements that are determined to be a practical way of meeting them. For example, a non-functional requirement to be free from backdoors may be satisfied by replacing it with a process requirement to use pair programming. Other non-functional requirements will trace to other system components and be verified at that level. For example system reliability is often verified by analysis at the system level. Avionics software with its complicated safety requirements must follow the DO-178B development process.

Requirements analysis or requirements engineering

Main article: Requirements analysis

Requirements are prone to issues of ambiguity, incompleteness, and inconsistency. Techniques such as rigorous inspection have been shown to help deal with these issues. Ambiguities, incompleteness, and inconsistencies that can be resolved in the requirements phase typically cost orders of magnitude less to correct than when these same issues are found in later stages of product development. Requirements analysis strives to address these issues.

There is an engineering trade off to consider between requirements which are too vague, and those which are so detailed that they

1. take a long time to produce - sometimes to the point of being obsolete once completed
2. limit the implementation options available
3. are costly to produce

Documenting requirements

Requirements are usually written as a means for communication between the different stakeholders. This means that the requirements should be easy to understand both for normal users and for developers. One common way to document a requirement is stating what the system must do. Example: 'The contractor must deliver the product no later than xyz date.' Other ways include use cases and user stories.

Changes in requirements

Requirements generally change with time. Once defined and approved, requirements should fall under change control. For many projects, requirements are altered before the system is complete. This is partly due to the complexity of computer software and the fact that users don't know what they want before they see it. This characteristic of

requirements has led to requirements management studies and practices.

Disputes regarding the necessity of rigour in software requirements

Most agile software development methodologies question the need for rigorously describing software requirements upfront, which they consider a moving target. Instead, extreme programming for example describes requirements informally using user stories (short summaries fitting on an index card explaining one aspect of what the system should do), and considers it the developer's duty to directly ask the customer for clarification. Agile methodologies also typically capture requirements in a series of automated acceptance tests.

See also

- Business requirements
- Software requirements
- Requirements analysis
- Requirements elicitation
- Requirements management
- Requirement prioritization
- Requirements traceability
- Specification (technical standard)
- MoSCoW Method - prioritisation technique

References

1. ^ Stellman, Andrew; Greene, Jennifer (2005). *Applied Software Project Management* (http://www.stellman-greene.com/aspm/) . O'Reilly Media. p. 98. ISBN 978-0-596-00948-9. http://www.stellman-greene.com/aspm/.
2. ^ Wiegers, Karl E. (2003). *Software Requirements, Second Edition*. Microsoft Press. ISBN 0-7356-1879-8.
3. ^ Young, Ralph R. (2001). *Effective Requirements Practices*. Addison-Wesley. ISBN 978-0201709124.
4. ^ Davis, Alan M. (1993). *Software Requirements: Objects, Functions, and States, Second Edition*. Prentice Hall. ISBN 0-13-805763-X.

External links

- *Discovering System Requirements* (http://www.sie.arizona.edu/sysengr/publishedPapers/ReqsSandiaReport.pdf)
- *The International Institute for Business Analysis and The IIBA's Business Analysis Body of Knowledge* (http://theiiba.org)
- *Writing Good Requirements* (http://www.reqid.com/requirements-management/writing-good-requirements-part1/)
- Gottesdiener, Ellen (2009). *The Software Requirements Memory Jogger: A*

Desktop Guide to Help Business and Technical Teams Develop and Manage Requirements (http://www.ebgconsulting.com/Pubs/srmj.php) . Addison-Wesley. ISBN 157681114X. http://www.ebgconsulting.com/Pubs/srmj.php.