Requirement

From Wikipedia, the free encyclopedia

In product development and process optimization, a **requirement** is a singular documented physical and functional need that a particular design, product or process must be able to 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 customer, organization, internal user, or other stakeholder. A **specification** (often abbreviated as **spec**) may refer to an explicit set of requirements *to be satisfied* by a material, design, product, or service.[1]

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.

Contents

Origins of term

The term **requirement** has been in use in the software engineering community since at least the 1960s.[2]

Product versus process requirements

Requirements can be said to relate to two fields:

- **Product requirements** prescribe properties of a system or product.
- **Process requirements** prescribe activities to be performed by the developing organization. For instance, process requirements could specify the methodologies that must be followed, and constraints that the organization must obey.

Product and process requirements are closely linked; a product requirement could be said to specify the automation required to support a process requirement while a process requirement could be said to specify the activities required to support 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).

Types of requirement

Requirements are typically classified into types produced at different stages in a development progression, with the taxonomy depending on the overall model being used.. For example, the following scheme was devised by the International Institute of Business Analysis in their Business Analysis Body of Knowledge[3] (see also FURPS and Types of requirements).

Architectural requirements

Architectural requirements explain what has to be done by identifying the necessary systems structure and systems behavior, i.e., systems architecture of a system.

Business requirements

High-level statements of the goals, objectives, or needs of an organization. They usually describe opportunities that an organization wants to realise or problems that they want to solve. Often stated in a business case.

User (stakeholder) requirements

Mid-level statements of the needs of a particular stakeholder or group of stakeholders. They usually describe how someone wants to interact with the intended solution. Often acting as a mid-point between the high-level business requirements and more detailed solution requirements.

Functional (solution) requirements

Usually detailed statements of capabilities, behaviour, and information that the solution will need. Examples include formatting text, calculating a number, modulating a signal. They are also known as *capabilities*.

Quality-of-service (non-functional) requirements

Usually detailed statements of the conditions under which the solution must remain effective, qualities that the solution must have, or constraints within which it must operate.[4] Examples include: reliability, testability, maintainability, availability. They are also known as *characteristics*, *constraints* or the *ilities*.

Implementation (transition) requirements

Usually detailed statements of capabilities or behaviour required only to enable transition from the current state of the enterprise to the desired future state, but that will thereafter no longer be required. Examples include: recruitment, role changes, education, migration of data from one system to another.

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.[5] [6]

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.
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 avoided.
Specify Importance	Many requirements represent a stakeholder-defined characteristic the absence of which will result in a major or even fatal deficiency. Others represent features that may be implemented if time and budget permits. The requirement must specify a level of importance.
Verifiable	The implementation of the requirement can be determined through basic possible methods: inspection, demonstration, test (instrumented) or analysis (to include validated modeling & simulation).

There are many more attributes to consider that contribute to the quality of requirements. If requirements are subject to rules of data integrity (for example) then accuracy/correctness and validity/authorization are also worthy attributes. Traceability confirms that the requirement set satisfies the need (no more - and no less than what is required).

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, 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

Requirements analysis or 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,[7] analysis (checking for consistency and completeness), specification (documenting the requirements) and validation (making sure the specified requirements are correct).[8][9]

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

Agile approaches evolved as a way of overcoming these problems, by baselining requirements at a high-level, and elaborating detail on a just-in-time or *last responsible moment* basis.

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 methods 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.

Issues

Competing standards

There are several competing views of what requirements are and how they should be managed and used. Two leading bodies in the industry are the IEEE and the IIBA. Both of these groups have different but similar definitions of what a requirement is.

The *Guide to the Business Analysis Body of Knowledge®* version 2 from IIBA defines a requirement as:

1. A condition or capability needed by a stakeholder to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a solution or solution component to satisfy a contract, standard, specification, or other formally imposed documents.
3. A documented representation of a condition or capability as in (1) or (2).[10]

This definition is based on IEEE 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology.[11]

Requirements creep

Scope creep may occur from requirements moving over time. In Requirements management the alteration of requirements is allowed but if not adequately tracked or preceding steps (business goals then user requirements) are not throttled by additional oversight or handled as a cost and potential program failure, then requirements changes are easy and likely to happen. It is easy for requirement changes to occur faster than developers are able to produce work, and the effort to go *backwards* as a result.

Multiple requirements taxonomies

There are multiple taxonomies for requirements depending on which framework one is operating under. (For example, the stated standards of IEEE, vice IIBA or U.S. DoD approaches). Differing language and processes in different venues or casual speech can cause confusion and deviation from desired process.

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. Some research suggests that software requirements are an illusion created by misrepresenting design decisions as requirements in situations where no real requirements are evident.[12]

Process corruptions

A process being run by humans is subject to human flaws in governance, where convenience or desires or politics may lead to exceptions or outright subversion of the process and deviations from the textbook way the process is supposed to proceed. Examples include:

- Process with no rigor gets no respect - If exceptions or changes are common, such as the organization running it having little independence or power or not being reliable and transparent in records, it may lead to the overall process being ignored.
- New players wanting a do-over - e.g., The natural tendency of a new person to want to change his predecessor's work to demonstrate his power or his claims of value, such as a new CEO wanting to change the previous CEO's planning, including business goals, of something (such as a software solution) already in development, or a newly created office objects to current development of a project because they did not exist when user requirements were crafted, so they begin an effort to backtrack and re-baseline the project.
- Coloring outside the lines - e.g., Users wanting more control do not just input things that meet the requirements management definition of "user requirement" or priority level, but insert design details or favored vendor characteristic as user requirements or everything their office says as the highest possible priority.
- Showing up late - e.g., Doing little or no effort in requirements elicitation prior to development. This may be due to thinking they will get the same benefit regardless of individual participation, or that there is no point if they can just insert demands at the testing stage and next spin, or the preference to be always right by waiting for post-work critique.

Within the U.S. Department of Defense process, some historical examples of requirements issues are

- the M-2 Bradley issues of casual requirements movement portrayed in Pentagon Wars;
- the F-16 growth from lightweight fighter concept of the Fighter mafia, attributed to F-15 program attempting to sabotage competition or individual offices putting in local desires eroding the concept of being lightweight and low cost.
- enthusiasm ca. 1998 for 'Net-Ready' led to its mandate as Key Performance Parameter from the Net-Ready office, outside the office defining requirements process and not consistent to that office's

previously defined process, their definition of what a KPP was, or that some efforts might not be appropriate or able to define what constituted 'Net-Ready'.

See also

- Business requirements
- Software requirements
- Requirements engineering
- Requirements analysis
- Requirements elicitation
- Requirements management
- Requirement prioritization
- Requirements traceability
- Specification (technical standard)
- Shall and will - phrasing
- MoSCoW Method - prioritisation technique
- User Story
- Use Case

References

1. ^ *Form and Style of Standards, ASTM Blue Book* (http://www.astm.org/COMMIT/Blue_Book.pdf). ASTM International. 2012. Retrieved 5 January 2013.
2. ^ Boehm, Barry (2006). "A view of 20th and 21st century software engineering" (http://dl.acm.org/citation.cfm?id=1134288). "ICSE '06 Proceedings of the 28th international conference on Software engineering". University of Southern California, University Park Campus, Los Angeles, CA: Association for Computing Machinery, ACM New York, NY, USA. pp. 12–29. ISBN 1-59593-375-1. Retrieved January 2, 2013.
3. ^ *A Guide to the Business Analysis Body of Knowledge® (BABOK® Guide) Version 2.0* (http://IIBA.org). 2009. ISBN 978-0-9811292-1-1.
4. ^ Ralph, P., and Wand, Y. A Proposal for a Formal Definition of the Design Concept. In, Lyytinen, K., Loucopoulos, P., Mylopoulos, J., and Robinson, W., (eds.), Design Requirements Engineering: A Ten-Year Perspective: Springer-Verlag, 2009, pp. 103-136
5. ^ Davis, Alan M. (1993). *Software Requirements: Objects, Functions, and States, Second Edition*. Prentice Hall. ISBN 0-13-805763-X.
6. ^ IEEE Computer Society (1998). *IEEE Recommended Practice for Software Requirements Specifications*. Institute of Electrical and Electronics Engineers, Inc. ISBN 0-7381-0332-2.
7. ^ 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.
8. ^ Wiegers, Karl E. (2003). *Software Requirements, Second Edition*. Microsoft Press. ISBN 0-7356-1879-8.

9. ^ Young, Ralph R. (2001). *Effective Requirements Practices*. Addison-Wesley. ISBN 978-0-201-70912-4.
10. ^ *A Guide to the Business Analysis Body of Knowledge® (BABOK® Guide) Version 2.0* (http://IIBA.org). 2009. p. 4. ISBN 978-0-9811292-1-1.
11. ^ "IEEE SA - 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology" (http://standards.ieee.org/findstds/standard/610.12-1990.html).
12. ^ Ralph, Paul (2012). "The Illusion of Requirements in Software Development" (http://link.springer.com/article/10.1007%2Fs00766-012-0161-4). *Requirements Engineering*.

External links

- *Discovering System Requirements* (http://prod.sandia.gov/techlib/access-control.cgi/1996/961620.pdf)

Look up ***requirement*** in Wiktionary, the free dictionary.

Retrieved from "http://en.wikipedia.org/w/index.php?title=Requirement&oldid=629910198"

Categories: Software requirements