On Non-Functional Requirements

Martin Glinz
Department of Informatics, University of Zurich, Switzerland
glinz@ifi.uzh.ch

Abstract

Although the term 'non-functional requirement' has been in use for more than 20 years, there is still no consensus in the requirements engineering community what non-functional requirements are and how we should elicit, document, and validate them. On the other hand, there is a unanimous consensus that non-functional requirements are important and can be critical for the success of a project.

This paper surveys the existing definitions of the term, highlights and discusses the problems with the current definitions, and contributes concepts for overcoming these problems.

1. Introduction

If you want to trigger a hot debate among a group of requirements engineering people, just let them talk about non-functional requirements. Although this term has been in use for more than two decades, there is still no consensus about the nature of non-functional requirements and how to document them in requirements specifications.

This paper is an attempt to work out and discuss the problems that we have with the notion of non-functional requirements and to contribute concepts for overcoming these problems. The focus is on system (or product) requirements; the role of non-functional requirements in the software process is not discussed [16].

The paper is organized as follows. Section 2 surveys typical definitions for the terms 'functional requirement' and 'non-functional requirement'. The problems with these definitions are discussed in Section 3. Section 4 presents concepts about how these problems can be overcome or at least alleviated. The paper ends with a discussion of these concepts.

2. Defining the term

In every current requirements classification (for example [7], [11], [12]), we find a distinction between requirements concerning the functionality of a system and other requirements.

There is a rather broad consensus about how to define the term *'functional requirements'*. The existing definitions follow two threads that coincide to a large extent. In the first thread, the emphasis is on functions: a functional requirement specifies "a function that a system (...) must be able to perform" [6], "what the product must do" [18], "what the system should do" [20]. The second thread emphasizes behavior: functional requirements "describe the behavioral aspects of a system" [1]; behavioral requirements are "those requirements that specify the inputs (stimuli) to the system, the outputs (responses) from the system, and behavioral relationships between them; also called functional or operational requirements." [3].

Wiegers as well as Jacobson, Rumbaugh and Booch try a synthesis: "A statement of a piece of required functionality or a behavior that a system will exhibit under specific conditions." [21]; "A requirement that specifies an action that a system must be able to perform, without considering physical constraints; a requirement that specifies input/output behavior of a system." [10].

There is only one semantic difference that may arise between the different definitions: timing requirements may be viewed as behavioral, while they are not functional. However, most publications in RE consider timing requirements to be performance requirements which in turn are classified as non-functional requirements.

On the other hand, there is no such consensus for *non-functional requirements*. Table 1 gives an overview of selected definitions from the literature or the web, which – in my opinion – are representative of the definitions that exist.

3. Where is the problem?

The problems that we currently have with the notion of non-functional requirements can be divided into *definition problems*, *classification problems* and *representation problems*.

1090-705X/07 $25.00 © 2007 IEEE
DOI 10.1109/RE.2007.45

Table 1. Definitions of the term *'non-functional requirement(s)'* (listed in alphabetical order of sources)

Source	Definition
Antón [1]	Describe the nonbehavioral aspects of a system, capturing the properties and constraints under which a system must operate.
Davis [3]	The required overall attributes of the system, including portability, reliability, efficiency, human engineering, testability, understandability, and modifiability.
IEEE 610.12 [6]	Term is not defined. The standard distinguishes design requirements, implementation requirements, interface requirements, performance requirements, and physical requirements.
IEEE 830-1998 [7]	Term is not defined. The standard defines the categories functionality, external interfaces, performance, attributes (portability, security, etc.), and design constraints. Project requirements (such as schedule, cost, or development requirements) are explicitly excluded.
Jacobson, Booch and Rumbaugh [10]	A requirement that specifies system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility, and reliability. A requirement that specifies physical constraints on a functional requirement.
Kotonya and Sommerville [11]	Requirements which are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet.
Mylopoulos, Chung and Nixon [16]	"... global requirements on its development or operational cost, performance, reliability, maintainability, portability, robustness, and the like. (...) There is not a formal definition or a complete list of nonfunctional requirements."
Ncube [17]	The behavioral properties that the specified functions must have, such as performance, usability.
Robertson and Robertson [18]	A property, or quality, that the product must have, such as an appearance, or a speed or accuracy property.
SCREEN Glossary [19]	A requirement on a service that does not have a bearing on its functionality, but describes attributes, constraints, performance considerations, design, quality of service, environmental considerations, failure and recovery.
Wiegers [21]	A description of a property or characteristic that a software system must exhibit or a constraint that it must respect, other than an observable system behavior.
Wikipedia: Non-Functional Requirements [22]	Requirements which specify criteria that can be used to judge the operation of a system, rather than specific behaviors.
Wikipedia: Requirements Analysis [23]	Requirements which impose constraints on the design or implementation (such as performance requirements, quality standards, or design constraints).

3.1. Definition problems

When analyzing the definitions in Table 1, we find not only terminological, but also major conceptual discrepancies. Basically, all definitions build on the following terms: *property* or *characteristic, attribute*, *quality*, *constraint*, and *performance*. However, there is no consensus about the concepts that these terms denote. There are also cases where the meaning is not clear, because terms are used without a definition or a clarifying example.

Property and *characteristic* seem to be used in their general meaning, i.e. they denote something that the system must have, which typically includes specific qualities such as usability or reliability, but excludes any functional quality. There is no consensus whether constraints also are properties: Jacobson, Booch and Rumbaugh [10] include them, others, e.g. Wiegers [21] and Antón [1] exclude them.

Attribute is a term that is used both with a broad and a narrow meaning. In IEEE 830-1998 [7], attributes are a collection of specific qualities, excluding performance and constraints. On the other hand, in the definition by Davis [3], every non-functional requirement is an attribute of the system.

Every requirement (including all functional ones) can be regarded as a *quality*, because, according to ISO 9000:2000 [8], quality is the "degree to which a set of inherent characteristics fulfils requirements". Similarly, every requirement can be regarded as a *constraint*, because it constrains the space of potential solutions to those that meet this requirement.

Hence, in all definitions that mention the term *quality*, its meaning is restricted to a set of specific qualities other than functionality: usability, reliability, security, etc.

Correspondingly, it is clear that *constraint* in the context of non-functional requirements must have a restricted meaning. However, there is no consensus among the existing definitions what precisely this restriction should be. For example, IEEE 830-1998 [7] or Jacobson, Booch and Rumbaugh [10] restrict the meaning of constraint to design constraints and physi-

cal constraints. Others, for example the definition in the Wikipedia article on requirements analysis [23], do not treat constraints as a sub-category of non-functional requirements, but consider every non-functional requirement to be a constraint. Davis [3] does not mention constraints at all when discussing non-functional requirements. Robertson and Robertson [18] have a rather specific view of constraints: they consider operational, cultural, and legal constraints to be non-functional properties, whereas design constraints are regarded as a concept that is different from both functional and non-functional requirements.

Performance is treated as a quality or attribute in many definitions. Others, e.g. IEEE 830-1998 [7] and Wikipedia [23] consider it as a separate category.

Another discrepancy exists in the *scope* of non-functional requirements. Some definitions emphasize that non-functional requirements have, by definition, a *global scope*: "global requirements" (Mylopoulos, Chung and Nixon [16]), "overall attributes" (Davis [3]). Accordingly, proponents of this global view separate functional and non-functional requirements completely. For example, in the Volere template [18] they are documented in two separate top-level sections. On the other hand, Jacobson, Booch and Rumbaugh [10] emphasize that there are both local non-functional requirements (e.g. most performance requirements) and global requirements such as security or reliability.

Most definitions refer to *system requirements* (also called *product requirements*) only and exclude *project* and *process requirements* either explicitly [7] or implicitly. However, in the definition by Kotonya and Sommerville [11], project requirements are considered to be non-functional requirements.

Finally, the variety and divergence of concepts used in the definitions of non-functional requirements also lead to definitions containing elements that are obviously misconceived. For example, the SCREEN glossary [19] lists failure and recovery as non-functional properties, while behavior in the case of failure and recovery from failures are clearly functional issues. Ncube [17] even defines non-functional requirements as "the behavioral properties...". Although this is more likely a severe typo than a misconception, the fact that this error went unnoticed in a PhD thesis illustrates the fuzziness of the current notions of non-functional requirements and the need for a clear and concise definition. The worst example is the definition in the Wikipedia article on non-functional requirements [22], which is so vague that it could mean almost anything.

3.2. Classification problems

When analyzing the definitions given in Table 1, we also find rather divergent concepts for sub-classifying non-functional requirements. Davis [3] regards them as qualities and uses Boehm's quality tree [2] as a sub-classification for non-functional requirements. The IEEE standard 830-1998 on Software Requirements Specifications [7] sub-classifies non-functional requirements into external interface requirements, performance requirements, attributes and design constraints, where the attributes are a set of qualities such as reliability, availability, security, etc. The IEEE Standard Glossary of Software Engineering Terminology [6] distinguishes functional requirements on the one hand and design requirements, implementation requirements, interface requirements, performance requirements, and physical requirements on the other. Sommerville [20] uses a sub-classification into product requirements, organizational requirements and external requirements.

More classification problems arise due to mixing three concepts that should better be separated. These are the concepts of *kind* (should a given requirement be regarded as a function, a quality, a constraint, etc.), *representation* (see below), and *satisfaction* (hard vs. soft requirements). For an in-depth discussion of this problem, see [5].

3.3. Representation problems

As long as we regard any requirement that describes a function or behavior as a functional requirement, the notion of non-functional requirements is *representation-dependent*. Consider the following example: A particular security requirement could be expressed as "The system shall prevent any unauthorized access to the customer data", which, according to all definitions given in Table 1 is a non-functional requirement. If we represent this requirement in a more concrete form, for example as "The probability for successful access to the customer data by an unauthorized person shall be smaller than 10^{-5}", this is still a non-functional requirement. However, if we refine the original requirement to "The database shall grant access to the customer data only to those users that have been authorized by their user name and password", we have a functional requirement, albeit it is still a security requirement. In a nutshell, the kind of a requirement depends on the way we represent it.

A second representational problem is the lack of consensus *where to document* non-functional requirements. As discussed above, some authors recommend the documentation of functional and non-functional requirements in separate chapters of the software requirements specification. The Volere template [18] is a prominent example for this documentation style. In IEEE 830-1998 [7], seven of the eight proposed SRS templates also separate the functional requirements

completely from the non-functional ones. On the other hand, when documenting requirements in a use-case-oriented style according to the Rational Unified Process [10], non-functional requirements are attached to use case as far as possible. Only the remaining (global) non-functional requirements are documented separately as so-called supplementary requirements.

However, there are also cases where a non-functional requirement affects neither a single functional requirement nor the system as a whole. Instead, it pertains to a specific set of functional requirements. For example, some subset of the set of all use cases may need secure communication, while the other use cases do not. Such a case cannot be documented adequately with classic requirements specification templates and must be handled by setting explicit traceability links. As we will see in the next chapter, aspect-orientation provides a solution for this problem.

4. Elements of a solution

4.1. A faceted classification

In [5], I have proposed that the classification problems, and – in a radical sense – also the definition problems be overcome by introducing a faceted classification for requirements (Fig. 1) where the terms 'functional requirement' and 'non-functional requirement' no longer appear.

Separating the concepts of *kind*, *representation*, *satisfaction*, and *role* has several advantages, for example: (i) It is the representation of a requirement (not its kind!) that determines the way in which we can verify that the system satisfies a requirement (Table 2). (ii) Decoupling the kind and satisfaction facets reflects the fact that functional requirements are not always hard and non-functional requirements not always soft. (iii) The role facet allows a clear distinction of (prescriptive) system requirements and (normative or assumptive) domain requirements.

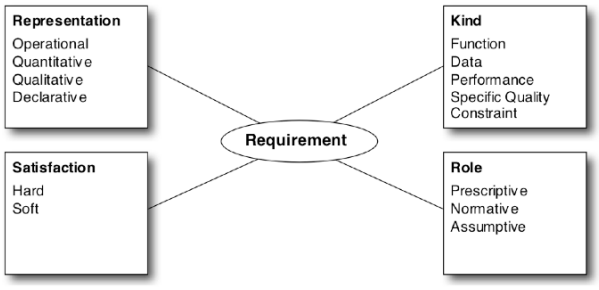

Figure 1. A faceted classification of requirements (from [5])

However, when using this classification it turned out that the idea of getting rid of the terms 'functional requirement' and 'non-functional requirement' is too radical. When practicing requirements engineering, *there is a need* to distinguish functional concerns from other, "non-functional" concerns and there is also a need for a sub-classification of the "non-functional" concerns in a clear and comprehensible way. In the next section, such a definition is presented.

Table 2. Representation determines verification [5]

Representation	Type of verification
Operational	Review, test or formal verification
Quantitative	Measurement (at least on an ordinal scale)
Qualitative	No direct verification. May be done by subjective stakeholder judgment of deployed system, by prototypes or indirectly by goal refinement or derived metrics
Declarative	Review

4.2. A definition based on concerns

We define a taxonomy of terms that is based on the concept of *concerns*, which makes it independent of the chosen representation. We assume a requirements engineering context, where *system* is the entity whose requirements have to be specified.

Furthermore, the taxonomy concentrates on *system requirements*. As project and process requirements are conceptually different from system requirements, they should be distinguished at the root level and not in a sub-category such as non-functional requirements.

DEFINITION. A *concern* is a matter of interest in a system. A concern is a *functional* or *behavioral* concern if its matter of interest is primarily the expected behavior of a system or system component in terms of its reaction to given input stimuli and the functions and data required for processing the stimuli and producing the reaction. A concern is a *performance* concern if its matter of interest is timing, speed, volume or throughput. A concern is a *quality concern* if its matter of interest is a quality of the kind enumerated in ISO/IEC 9126 [9].

DEFINITION. The set of all requirements of a system is partitioned into *functional requirements*, *performance requirements*, *specific quality requirements*, and *constraints*.

A *functional requirement* is a requirement that pertains to a functional concern.

A *performance requirement* is a requirement that pertains to a performance concern.

A *specific quality requirement* is a requirement that pertains to a quality concern other than the quality of meeting the functional requirements.

A *constraint* is a requirement that constrains the solution space beyond what is necessary for meeting

the given functional, performance, and specific quality requirements.

An *attribute* is a performance requirement or a specific quality requirement.

The taxonomy defined above is visualized in Figure 2. In my opinion, this structure is sufficient and useful both for theoretical and practical work. For persons who do not want to dispose of the term 'non-functional requirement', we can define this term additionally as follows.

DEFINITION. A *non-functional requirement* is an attribute of or a constraint on a system.

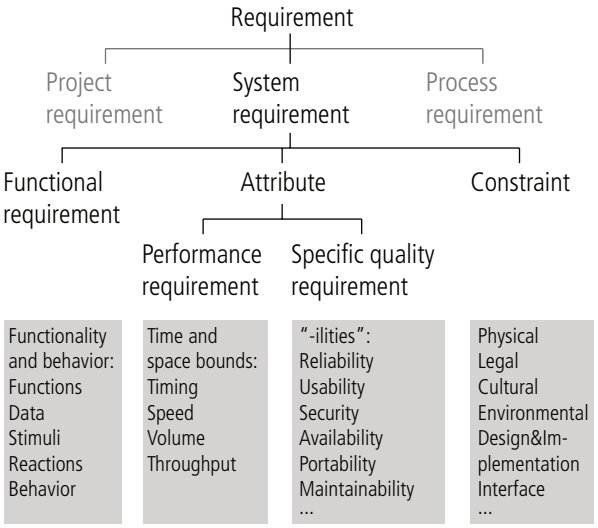

Figure 2. A concern-based taxonomy of requirements

Performance is a category of its own in our taxonomy (as well as in the IEEE standards [6] [7]) because performance requirements are typically treated separately in practice. This is probably due to the fact that measuring performance is not difficult a priori, while measuring other attributes is: there is a broad consensus to measure performance in terms of time, volume, and volume per unit of time. For any other attribute, there are no such generally agreed measures, which means that the task of eliciting specific qualities always implies finding an agreement among the stakeholders how to measure these qualities.

Interfaces, which are a separate category in the IEEE standards [6] [7], no longer appear in the terminology defined above. An interface requirement is classified according to the concern it pertains to as functional, performance, specific quality, or constraint.

Due to the systematic construction of our taxonomy, the task of *classifying* a given requirement becomes easier and less ambiguous. The often heard rule 'What the system does → functional requirement; How the system behaves → non-functional requirement' [4] is too coarse and leads to mis-classifications when attributes and constraints are represented functionally or when they are very important[1].

Table 3 gives the classification rules for the taxonomy laid out in Figure 2.

Table 3. Classification rules

No[1]	**Question** Was this requirement stated because we need to specify...	**Result**
1	... some of the system's behavior, data, input, or reaction to input stimuli – regardless of the way how this is done?	Functional
2	... restrictions about timing, processing or reaction speed, data volume, or throughput?	Performance
3	... a specific quality that the system or a component shall have?	Specific quality
4	... any other restriction about what the system shall do, how it shall do it, or any prescribed solution or solution element?	Constraint

[1] Questions must be applied in this order

4.3. Aspect-oriented representation

As soon as we structure the functional requirements specification systematically (by using a text template or by modeling requirements), the question about structuring the non-functional requirements comes up. Structuring attributes and constraints into sub-kinds according to a documentation template is helpful here, but clearly this is not enough; in particular when we have attributes and constraints that are neither completely local nor fully global, but pertain to some specific parts of the system.

An aspect-oriented representation of requirements, in particular of attributes and constraints, helps overcome this problem. A multi-dimensional separation of concerns [15] allows every concern to be modeled separately. Thus, all concerns are treated equally, which looks clean and elegant from an academic viewpoint. However, in practice, there is almost always a dominant concern, which is typically a functional one. This concern is crosscut by other concerns, both functional and non-functional ones.

In my research group, we have developed an aspect-oriented-extension for a hierarchical modeling lan-

[1] For example, in particle physics, the detectors of contemporary accelerators produce enormous amounts of data in real time. When asked a 'what shall the system do' question about the data processing software for such a detector, one of the first answers a physicist typically would give would be that the system must be able to cope with the data volume. However, this is a performance requirement.

guage [13], [14] where we identify a dominant functional concern and decompose the system model hierarchically according to the structure of this concern. All other concerns are modeled as aspects of this primary model. Aspect composition is supported by formal model weaving semantics.

Thus we can document attributes and constraints as separate entities, but, at the same time, attach them systematically to those elements in the primary model hierarchy where they apply. Global attributes and constraints are attached to the root of the decomposition hierarchy, while an attribute or constraint that restricts only some parts of the model is attached exactly to these parts by modeling join relationship from the aspect to the affected parts of the primary model.

5. Discussion

The analysis of the definitions given in Table 1 reveals the deficiencies of the current terminology. The *new definitions* proposed in this paper

- are more systematically constructed than any existing definitions that I am aware of,
- are representation-independent; i.e. the kind of a requirement is always the same, regardless of its representation,
- make it easier to classify a given requirement: with the classification criteria given in Table 3, the classification is much less ambiguous than with traditional definitions,
- better support the evolution of a requirements specification, because the classification of a requirement remains invariant under refinement and change as long as the concern to which the requirements pertains remains the same.

With an aspect-oriented documentation of attributes and constraints, the *documentation* and *traceability* problems of non-functional requirements are alleviated.

As this is mainly theoretical work, the *validation* of the theoretical soundness and usefulness of these ideas will be the extent to which other researchers find these ideas useful and adopt or build upon them.

The systematic exploration of the *practical usefulness* is a topic for further investigation.

References

[1] A. Antón (1997). *Goal Identification and Refinement in the Specification of Information Systems*. PhD Thesis, Georgia Institute of Technology.

[2] B. Boehm et al. (1976). Quantitative Evaluation of Software Quality. *Proc. 2nd IEEE International Conference on Software Engineering*. 592-605.

[3] A. Davis (1993). *Software Requirements: Objects, Functions and States*. Prentice Hall.

[4] X. Franch (1998). Systematic Formulation of Non-Functional Characteristics of Software. *Proc. 3rd Int'l Conf. Requirements Engineering (ICRE'98)*. 174-181.

[5] M. Glinz (2005). Rethinking the Notion of Non-Functional Requirements. *Proc. Third World Congress for Software Quality (3WCSQ 2005)*, Munich, Germany, Vol. II. 55-64.

[6] IEEE (1990). *Standard Glossary of Software Engineering Terminology*. IEEE Standard 610.12-1990.

[7] IEEE (1998). *IEEE Recommended Practice for Software Requirements Specifications*. IEEE Std. 830-1998.

[8] ISO 9000 (2000). *Quality Management Systems – Fundamentals and Vocabulary*. International Organization for Standardization.

[9] ISO/IEC 9126-1 (2001). *Software Engineering – Product Quality – Part 1: Quality Model*. International Organization for Standardization.

[10] I. Jacobson, G. Booch, and J. Rumbaugh (1999). *The Unified Software Development Process*. Reading, Mass.: Addison Wesley.

[11] G. Kotonya, I. Sommerville (1998). *Requirements Engineering: Processes and Techniques*. John Wiley & Sons.

[12] A. van Lamsweerde (2001). Goal-Oriented Requirements Engineering: A Guided Tour. *Proc. 5th International Symposium on Requirements Engineering (RE'01)*, Toronto. 249-261.

[13] S. Meier, T. Reinhard, C. Seybold, and M. Glinz (2006). Aspect-Oriented Modeling with Integrated Object Models. *Proc. Modellierung 2006*, Innsbruck, Austria. 129-144.

[14] S. Meier, T. Reinhard, R. Stoiber, M. Glinz (2007). Modeling and Evolving Crosscutting Concerns in ADORA. *Proc. Early Aspects at ICSE: Workshop in Aspect-Oriented Requirements Engineering and Architecture Design*.

[15] A. Moreira, A. Rashid, and J. Araújo (2005). Multi-Dimensional Separation of Concerns in Requirements Engineering. *Proc. 13th IEEE International Requirements Engineering Conference (RE'05)*. 285-296.

[16] J. Mylopoulos, L. Chung, B. Nixon (1992). Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. *IEEE Transactions on Software Engineering* **18**, 6 (June 1992). 483-497.

[17] C. Ncube (2000). *A Requirements Engineering Method for COTS-Based Systems Development*. PhD Thesis, City University London.

[18] S. Robertson and J. Robertson (1999). *Mastering the Requirements Process*. ACM Press.

[19] SCREEN (1999). *Glossary of EU SCREEN Project*. http://cordis.europa.eu/infowin/acts/rus/projects/screen/glossary/glossary.htm (visited 2007-07-05)

[20] I. Sommerville (2004). *Software Engineering*, Seventh Edition. Pearson Education.

[21] K. Wiegers (2003). *Software Requirements*, 2nd edition. Microsoft Press.

[22] Wikipedia: *Non-Functional Requirements* http://en.wikipedia.org/wiki/Non-functional_requirements (visited 2007-07-05)

[23] Wikipedia: *Requirements Analysis* http://en.wikipedia.org/wiki/Requirements_analysis (visited 2007-07-05)