Rethinking the Notion of Non-Functional Requirements

Martin Glinz

Department of Informatics, University of Zurich
Winterthurerstrasse 190, CH 8057 Zurich, Switzerland
glinz@ifi.unizh.ch
http://www.ifi.unizh.ch/~glinz

Abstract. Requirements standards and textbooks typically classify requirements into *functional requirements* on the one hand and *attributes* or *non-functional requirements* on the other hand. In this classification, requirements given in terms of required operations and/or data are considered to be functional, while performance requirements and quality requirements (such as requirements about security, reliability, maintainability, etc.) are classified as non-functional.
In this paper, we present arguments why this notion of non-functional requirements is flawed and present a new classification of requirements which is based on four facets: *kind* (e.g. function, performance, or constraint), *representation* (e.g. operational, quantitative or qualitative), *satisfaction* (hard or soft), and *role* (e.g. prescriptive or assumptive). We define the facets, discuss typical combinations of facets and argue why such a faceted classification of requirements is better than the traditional notion of functional and non-functional requirements.

1 Introduction

Quality, as defined by ISO 9000:2000 [8], is the "degree to which a set of inherent characteristics fulfils requirements", where a requirement is a "need or expectation that is stated, generally implied or obligatory". Hence, quality and requirements are closely intertwined concepts.

A lot of effort has been put into classifying qualities, the quality models by Boehm [2], McCall and Matsumoto [12] and ISO/IEC [9] being the best known ones. For example, the ISO/IEC 9126 model classifies qualities at the top level into functionality, reliability, usability, efficiency, modifiability, and portability.

Analogously, there have been many efforts to classify requirements and to establish links between qualities and requirements. In every current requirements classification, we find a distinction between *functional* and *non-functional* requirements, for example [6], [10], [11]. Davis [3] makes the same distinction, but calls them behavioral vs. non-behavioral requirements. Functional requirements are defined as those requirements that "describe what the system should do" [14], while all other requirements are considered to be non-functional.

However, there is no consensus, and it is in fact not clear, what a non-functional requirement really is. Firstly, we 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 830-1993 on Software Requirements Specifications [6] 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 [5] distinguishes functional requirements on the one hand and design requirements, implementation requirements, interface requirements, performance requirements, and physical requirements on the other hand. The term 'non-functional requirement' is not listed in this glossary. Sommerville [14] uses a sub-classification into product requirements, organizational requirements and external requirements. Some people consider goals and non-functional requirements to be more or less synonymous.

Secondly, in the terminology introduced above, the same requirement can be considered to be functional or non-functional, depending on the way we represent it. Consider the following example [10]: A particular security requirement could be expressed as "Any unauthorized access to the customer data shall be prevented", which would be classified as a non-functional requirement. If we represent this requirement in a more concrete form, for example as "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. As this requirement remains to be a security requirement, some people regard it still as a non-functional requirement which is just represented in a functional form.

Thirdly, we have the problem of the so-called soft requirements. A soft requirement is one which has a soft satisfaction criterion, i.e. satisfaction can range on a scale from weakly satisfied to fully satisfied. Customers tend to consider soft requirements and non-functional requirements to be synonymous concepts. However, this is a flawed conceptualization. For example, response time requirements in a user interface are typically soft, whereas response time requirements in real-time systems can be hard.

From these examples it becomes obvious that the distinction of functional vs. non-functional requirements and the various sub-classifications of the latter are a fuzzy classification concept. At closer examination, we discover that this fuzziness is rooted in the fact that the traditional classifications mix three different concepts: a classification according to *kind* (e.g. whether a requirement concerns a function, a performance need, a constraint, etc.), another according to soft or hard *satisfaction* and yet another one according to *representation* (e.g. whether a requirement is represented operationally, quantitatively or qualitatively).

In this paper, we present a new faceted classification of requirements which overcomes the fuzziness of traditional classifications by classifying requirements separately according to the three facets of *kind*, *satisfaction*, and *representation*. Moreover, we add the facet of *role*, an issue that is rather important in requirements engineering, but is often neglected.

The remainder of this paper is organized as follows. In Section 2, we introduce our faceted classification and define the facets. This is the main section of the paper. Section 3 discusses typical combinations of facet values and maps them to other classifications. Section 4 summarizes and concludes the paper.

2 A new faceted classification of requirements

2.1 What and why

As sketched in the introduction, we can classify requirements according to four different criteria. Which classification(s) we choose, depends on the purpose of the classification:

- In most cases, we want to classify requirements according to their *kind*, in order to differentiate between, for example, requirements that describe a required function or that specify a required performance behavior.
- When representing requirements, we need to differentiate between forms of *representation* such as an operational or a quantitative form.
- When it comes to the question of requirements *satisfaction*, we must differentiate between hard and soft requirements.
- Finally, we frequently have to distinguish requirements about a system-to-be from both factual and assumptive requirements about its environment. This is a classification according to the *role* that a requirement plays.

Figure 1 unifies these four classification aspects into a faceted classification of requirements. In the next four sub-sections, we describe the facets in more detail.

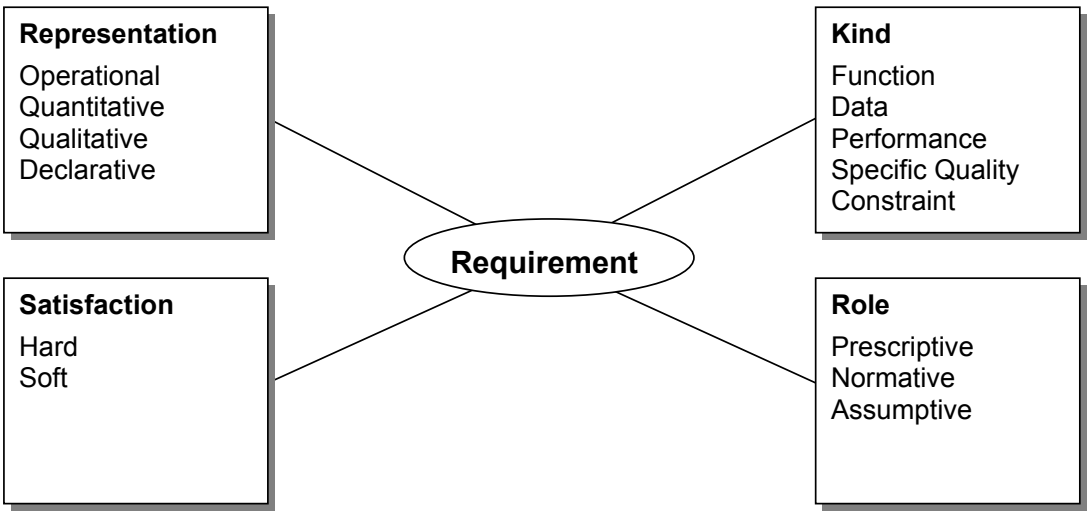

Fig. 1. A faceted classification of requirements

2.2 The kind facet

Requirements can be classified according to their kind, i.e. the matter that they concern. Firstly, a requirement can pertain to a *function* or to *data*; this is what we traditionally call functional requirements. Secondly, a requirement can specify *performance* in terms of speed, volume, throughput, etc. Thirdly, a requirement may specify a *specific quality* such as reliability or portability. Finally, a requirement may be a *constraint*, i.e. a design decision or design constraint imposed by stakeholders of

the system-to-be. As such decisions and constraints come from stakeholders, they are true requirements. Table 1 summarizes the classes of this classification facet.

Table 1. Requirements classified according to their kind

Kind	Definition
Function	A function that a system shall perform
Data	A data item or data structure that shall be part of a system's state
Performance	A requirement pertaining to time (points in time, reaction time, time intervals), speed, volume, or rates (volume per time unit)
Specific quality	Qualities concerning both properties of product *use* (e.g. reliability, usability) and product *management* (e.g. maintainability or portability)
Constraint	A design decision or design constraint imposed by a stakeholder (i.e. an item which is not under control of the system designers)

2.3 The representation facet

There are four typical forms for representing requirements. As the representation form goes hand in hand with the way how a requirement can be verified, it makes sense to classify requirements according to this form.

Requirements that describe what the system-to-be is supposed to do (those pertaining to functions, see above) are typically represented in an *operational* form: actions to be performed, data to be provided, states to be entered, etc. Requirements of this type are verified by reviewing, testing or by formal verification.

Performance requirements must be specified in a *quantitative* form if we want these requirements to be precise, unambiguous and verifiable. The same is true for qualities such as availability or reliability. Quantitatively specified requirements are verified by measuring (on a scale which is at least ordinal).

However, in particular on more abstract levels, we also want to state requirements in a *qualitative* form, for example when stating business goals ("The system shall simplify the order tracking process") or usability goals ("The system shall be easy to use for casual users"). Such goals can't be verified directly. Only after deployment of the system or with the help of prototypes, stakeholders can experience and subjectively judge whether or not a qualitative requirement is satisfied.[1]

Finally we have requirements that just describe some required situation. This is typically the case with data and with constraints (e.g. "The system shall run on a Linux platform"). We call this form of representation *declarative*. The typical way of verifying such requirements is by reviewing.

Table 2 summarizes the definition and the type of verification of the four representation forms.

[1] Alternatively, a quantitative requirement can be verified *indirectly* by decomposing it or by deriving metrics that (hopefully) are highly correlated with the given requirement, for example by applying GQM [1] or Gilb's measurement approach [4]. The NFR-framework developed by Mylopoulos et al. [13] is a typical example of a goal decomposition approach, where a goal is refined into sub-goals and design decisions. Within the NFR-framework, one can reason about the satisficability of a goal by recursively determining the satisficability of the sub-goals and considering the relationships between the sub-goals and the goals.

Table 2. A representation-based classification of requirements

Form	Definition	Type of verification
Operational	Specification of operations or data	Review, test or formal verification
Quantitative	Specification of measurable properties	Measurement (at least on an ordinal scale)
Qualitative	Specification of goals	No direct verification. Either by subjective stakeholder judgment of deployed system, by prototypes or indirectly by goal refinement or derived metrics (see footnote on previous page)
Declarative	Description of a required feature	Review

2.4 The satisfaction facet

Verification of requirements means that we have to determine whether the chosen solution satisfies the requirements. When we examine the criteria to be used for deciding whether a requirement is satisfied, we find two cases: in the first case, a requirement is either completely satisfied or it is not satisfied. Requirements of this kind are called *hard*. In the second case, a requirement can be gradually satisfied, that means the degree of satisfaction is measured on a scale which is at least ordinal. In this case, we speak of *soft* requirements [7].

Let us assume that the value of a solution feature is proportional to the degree to which it satisfies its requirements. When we plot this value over the effort for designing and implementing a solution feature, we get a 0-1 curve for hard requirements (Fig. 2a) and an S shaped, continuously growing curve for soft requirements (Fig. 2b). Hence, it makes sense for soft requirements to have both a planned degree of satisfaction and a minimum acceptable degree of satisfaction [4].

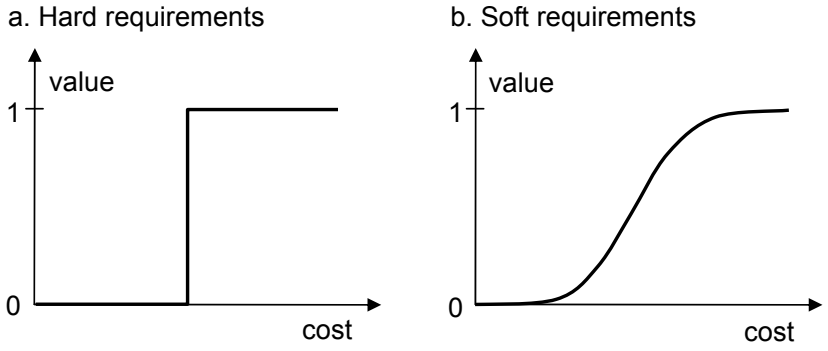

Fig. 2. Cost/value curves for (a) hard and (b) soft requirements

2.5 The role facet

A requirement can play three *roles* in a requirements specification: (1) it can specify properties of the system-to-be, (2) it can state facts or rules in the system environment that influence the design and implementation of the system-to-be, and (3) it can

specify how an actor in the system environment should behave when interacting with the system.

The requirements in the first role are the "classic" requirements that concern only the system-to-be. We call these requirements *prescriptive*.

We call requirements in the second role *normative* requirements, because they typically describe norms in the system environment that the system must be aware of [15]. For example, if we specify a system that shall process income tax forms and compute income taxes, we have normative requirements that describe the tax computation formulae, rules for deductibles, etc. which are given by tax laws and decrees.

The requirements in the third role describe behavior of actors that the system-to-be can't control. For example, in an information system that supports dispatching of ambulances, we have a requirement that the dispatcher has to react to an incoming emergency message by sending an ambulance. The system can only control whether the dispatcher acknowledges the message within a given interval of time, but it can't control if she or he actually took the appropriate action. Such requirements specify how an actor should behave; hence we call them *assumptive* requirements.

2.6 Examples

Table 3 shows how a couple of sample requirements are classified in our new classification scheme.

Table 3. Sample requirements with their classification

Requirement	Classification
"The system shall compute the sum of all applicable deductions."	Kind: function Representation: operational Satisfaction: hard Role: prescriptive
"The applicable deductions are computed according to the formula D = <fff>, which is stated in income tax decree No. <nnn> of <ddmmyy>." [with concrete values filled in for placeholders <fff>, <nnn>, and <ddmmyy> in a real situation]	Kind: function Representation: operational Satisfaction: hard Role: normative
"The system shall be easy to use by casual users."	Kind: specific quality Representation: qualitative Satisfaction: soft Role: prescriptive
"The response time shall be less than 1 s on average"	Kind: performance Representation: quantitative Satisfaction: soft Role: prescriptive
"The system shall run on PCs featuring at least a 500MHz CPU and 256MB main memory."	Kind: constraint Representation: quantitative Satisfaction: soft Role: prescriptive
"The user must provide accurate data for all input fields of the form."	Kind: data Representation: declarative Satisfaction: hard Role: assumptive

3. Facet combinations and dependencies

In this section, we describe some typical combinations of facets and map them to concepts in other classifications used in practice and in the literature.

3.1 Typical combinations

Some combinations of facet values occur more frequently than others and constitute typical sorts of requirements found in practice and in the literature. For example, the classic notion of a functional requirement is (function or data, operational or declarative, hard, prescriptive or undetermined), where 'prescriptive or undetermined' means that such a requirement typically is either prescriptive or the role facet is undetermined. Table 4 lists some typical combinations.

Table 4. Some typical facet combinations

Traditional Classification	**Faceted Classification**			
	Kind	**Representation**	**Satisfaction**	**Role**
"Classic" functional requirement	Function	Operational	Hard	Prescriptive or undetermined
	Data	Declarative	Hard	Prescriptive or undetermined
Good (i.e. quantified) performance or quality requirement	Performance or specific quality	Quantitative	Soft	Prescriptive or undetermined
a. Goal b. Bad performance or quality requirement	Performance or specific quality	Qualitative	Soft	Prescriptive or undetermined
"Classic" constraint	Constraint	Declarative	Hard	Prescriptive
"Classic" goal	Any except constraint	Any	Soft	Prescriptive or assumptive
Softgoal [13]	Specific quality	Qualitative (also quantitative or operational)	Soft	Prescriptive or assumptive
–	Function	Operational	Soft	Prescriptive
–	Performance	Quantitative	Hard	Prescriptive
–	Function	Operational	Hard	Normative or assumptive
–	Data or Constraint	Declarative	Hard	Normative or assumptive

Table 4 demonstrates once again that, with a faceted classification, we also can characterize requirements beyond the traditional taxonomy of functional vs. non-functional requirements. For example, requirements with functional kind and operational representation are traditionally assumed to be hard. However, this is not always the case. The requirement "The system shall monitor all essential events" concerns a

function (monitoring) and is represented operationally, but "all essential events" typically will have a range of satisfaction, thus yielding a soft requirement.

As another example, in real-time systems we typically have both hard and soft performance requirements, while in a traditional view, performance requirements are non-functional requirements, which in turn are considered to be soft.

On the other hand, we also have combinations that never or almost never occur, for example, a function is never stated in quantitative form.

3.2 Goals vs. requirements

Goal-driven approaches [11], [13] advocate a hierarchical AND/OR decomposition of goals. Goals are ultimately refined into operationalized requirements. Upper level goals in such goal decompositions are typically classified as (specific quality, qualitative, soft, –), where "–" means that no value has been assigned to the role facet yet. On medium and lower goal levels, one would also use 'function', 'data' or 'performance' in the kind facet, and 'declarative' in the representation facet. On the lowest level, where everything is operationalized, we have a classification of (function or data, operational, hard, –). When such lowest level goals are assigned to actors in the system or in the environment, the role facet is correspondingly set to 'prescriptive' or 'assumptive'.

4 Conclusions

Summary. In this paper, we have rethought the classic notion of non-functional requirements. We have demonstrated that this is a fuzzy concept, because it mixes elements that should be considered separately. In order to do this separation properly, we have defined a new faceted classification which distinguishes four facets: kind, representation, satisfaction and role.

Benefits. The traditional notion of non-functional requirements is a fuzzy concept: something that has to do with quality, somehow soft, mostly qualitative, but better quantitative (or even better operationalized?), etc. Our new classification has the benefit of *separating concerns* clearly, which allows us to characterize a requirement much more precisely. For example, we avoid the problem that a requirement is considered to be non-functional if it is stated in qualitative form, while the same requirement becomes a functional one when stated operationally.

The representation facet of the classification helps us *identify the proper type of verification* for a given requirement – as this is primarily determined by the way how a requirement is represented. For example, an operational requirement will typically be verified by acceptance testing, while a quantitative requirement is verified by measurement. Moreover, the satisfaction facet provides information whether the verification is *discrete* (i.e. either completely satisfied or not satisfied), or whether we have a *range* of acceptable behavior.

As another advantage, the classification provides us with *precision and completeness criteria*: the more qualitative requirements we have, the less precise and (in an evolutionary sense) the less complete a requirements specification is.

Finally, the role facet helps us shaping (and sharpening) the *boundary* between the *system* and its *environment*.

Related work. Some other classifications, in particular the traditional ones, have been described in the introduction.

Van Lamsweerde [11] uses another classification with a restricted notion of requirements. He uses goals (which are objectives of a composite system), requisites (goals that are controllable by some individual agent), requirements (requisites assigned to system agents) and assumptions (requisites assigned to environment agents). Requisites must eventually be operationalized. All concepts can be expressed by a corresponding combination of facet values in our classification; for example, an operationalized assumption would be classified as operational in the representation facet and assumptive in the role facet. It will most probably concern a function and be hard with respect to satisfaction.

Mylopoulos et al. [13] have defined the so-called NFR (non-functional requirements) framework. It is based on softgoals which are structured hierarchically with AND/OR graphs. Softgoals are soft requirements in our classification. Softgoals may be stated qualitatively, quantitatively, or in operationalized form, which is covered by our representation facet.

Gilb [4] uses a classification with two facets: functions, qualities, costs, and constraints on the one hand and quantifiable vs. non-quantifiable on the other hand. In his classification, functions are always non-quantifiable, while qualities and costs are quantifiable. Constraints may be of either kind. Gilb believes that qualities and costs always should be quantified and that qualitative statements such as “high usability” constitute ‘false’ requirements. He does not distinguish hard and soft requirements. However by stating that functions tend to be either present or absent, he makes clear that he considers functions to be hard requirements. Furthermore, he distinguishes two satisfaction levels for quantifiable requirements: “must” (the lowest acceptable level) and “plan” (the planned level). However, our classification reveals that this distinction is in fact a feature of soft requirements, not of quantitatively represented ones: there exist quantitatively represented requirements which are hard, i.e. they are either completely satisfied or not satisfied at all. In this case, a distinction of “must” and “plan” does not make sense.

We are not aware of any other comprehensive approach to the problem of requirements classification.

Open issues and next steps. While the new classification is clearly better than traditional ones from a conceptual standpoint (see above), it has yet to stand the test of practical usefulness in the daily life of requirements engineering.

From a research standpoint, experience from practical application will have to be evaluated, in particular with respect size vs. usefulness: should the classification be simplified, or on the contrary, should it be extended by additional facets (a priority facet, for example).

Acknowledgements

I thank Jochen Ludewig (University of Stuttgart) for his valuable comments on the classification concept presented in this paper.

References

1. Basili, V., G. Caldiera, D. Rombach (1994). Goal Question Metric Paradigm. In J. J. Marciniak (ed.): *Encyclopedia of Software Engineering* **1**. New York: John Wiley&Sons. 528-532.
2. Boehm B. et al. (1976). Quantitative Evaluation of Software Quality. *Proceedings of the 2nd IEEE International Conference on Software Engineering*. 592-605.
3. Davis, A. (1992). *Software Requirements: Objects, Functions and States*. Prentice Hall.
4. Gilb, T. (1997). Towards the Engineering of Requirements. *Requirements Engineering* **2**, 3 165-169.
5. IEEE (1990). *Standard Glossary of Software Engineering Terminology*. IEEE Standard 610.12-1990.
6. IEEE (1993). *IEEE Recommended Practice for Software Requirements Specifications*. IEEE Standard 830-1993.
7. Irvine, C., T. Levin (2000). Quality of Security Service. *Proceedings of the 2000 Workshop on New Security Paradigms*. 91-99.
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. Kotonya, G., I. Sommerville (1998). *Requirements Engineering: Processes and Techniques*. John Wiley & Sons.
11. Van Lamsweerde, A. (2001). Goal-Oriented Requirements Engineering: A Guided Tour. *Proceedings of the 5th International Symposium on Requirements Engineering (RE'01),* Toronto. 249-261.
12. McCall, J.A., Matsumoto, M.T. (1980). *Software Quality Measurement Manual, Vol. II.* Rome Air Development Center, RADC-TR-80-109-Vol-2.
13. Mylopoulos, J., 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.
14. Sommerville, I. (2004). *Software Engineering*, Seventh Edition. Pearson Education.
15. Wieringa, R.J. (2000). The Declarative Problem Frame: Designing Systems that Create and Use Norms. *Proceedings of the Tenth International Workshop on Software Specification and Design*. San Diego. 75-85.