# Requirements Classification as a First Step to Grasp Quality Requirements

**Elke Hochmüller**

Institut für Informatik
Universität Klagenfurt, Austria
elke@ifi.uni-klu.ac.at

## Abstract

As the degree of compliance of quality requirements mainly induces the acceptance and success of a system, special emphasis should be laid on the elicitation, specification and validation of quality requirements. Current requirements definition methods, however, mainly focus on objects, functions, and states. This paper proposes an integrated treatment of requirements (functional as well as quality and economic requirements) based on a faceted and adaptable classification approach. Possible impact and relevance of this classification strategy will be outlined.

## 1 Motivation

Requirements analysis turned out to be the most crucial task in software engineering. In order to support the various activities of requirements elicitation, specification, and validation a broad spectrum of methods, techniques, and tools have been proposed, developed and used so far. Most of the approaches used for requirements modelling concentrate on structuring and representing functionality. But functionality is not the only one dimension that matters in software development (c.f. [21,24]). Cancelled projects, unprofitable products, unhappy users, and budget and schedule overruns are some of the symptoms that arise from requirements which were not taken into account properly [18]. In fact, there exists a dimension of requirements which can be regarded as the root for those difficulties - in the literature that is referred to as non-functional requirements or (software) quality requirements.

The deficiencies of giving quality requirements proper treatment can mainly be traced back to inadequate techniques to grasp as well as to express them in an appropriate manner because of their rather informal nature. Hence, no wonder that as a consequence quality requirements are often simply forgotten or even neglected during analysis. They then will turn up again far too late during acceptance test or even during operation, but quality cannot be attached to a product a posteriori [2]. The approach presented in the sequel provides an integrated view of requirements with special focus on quality requirements.

The intention of this paper is not only to provide a clear starting point to characterize quality requirements but also to emphasize the necessity to incorporate them inherently into the software development process. The seriousness of the problems connected with that dimension of requirements becomes obvious in the scientific community's inability to even agree upon its existence not to mention its classification. In discussing these facts, the need for a proper treatment of quality requirements will be motivated subsequently.

A generic classification structure which emphasizes on quality requirements and their embodiment will be proposed. This classification approach takes into account that a final and complete characterization of quality requirements is not possible at the moment. It assures that the extensibility of the characterization framework may not be hindered from the beginning in order to not fare alike previous characterization attempts on quality requirements. Moreover, the proposed classification scheme together with an adequate information system can contribute to achieve better documented requirements of a higher quality and support elicitation, communication, traceability, and control activities.

Nevertheless, an organizational commitment in favour of quality requirements is needed. Anchoring them explicitly in the system development process will be an inevitable necessity.

## 2 Non-functional requirements - spook or reality?

In the literature there exist numerous attempts in defining non-functional requirements. One of the first characterization approaches dates back to 1976 when Boehm introduced a hierarchical list of quality attributes [3]. His recent work [4] includes the mapping of stakeholder's primary concerns onto these quality criteria in order to get a better starting point for negotiations in case of conflicting requirements. Davis [9] prefers the notions of behavioural and non-behavioural requirements and focusses primarily on reliability and efficiency issues. The IEEE-Std. 830 [12] defines non-functional requirements in terms of performance, external interfaces, design constraints, and quality attributes. In addition to functionality, the ISO 9126 Standard [13] distinguishes five classes of non-functional requirements as main characteristics for evaluation of product quality, namely reliability, usability, efficiency, maintainability, and portability. Roman [26] classifies nonfunctional requirements according to interface constraints, performance constraints, operating constraints, life cycle constraints, economic constraints, and political constraints. A methodology developed by the Rome Air Development Center (RADC) considers non-functional requirements from two perspectives and distinguishes between user-oriented software quality factors (e.g. efficiency, correctness, interoperability) and system-oriented software quality criteria (e.g. completeness, anomaly management, functional scope) [16].

All approaches mentioned above address quality of and constraints on a particular *software product* - they are so-called product-centered approaches. Less attention, however, is paid to the quality of and constraints on the *software development process* on its own. Only few approaches address also requirements upon the development process, like those in [20] and [30] distinguishing between product requirements, process requirements, and external requirements, which can be legal or economic constraints placed on both product and process.

From all these different definitions and recent discussions (c.f. REFSQ'94 [31], 8th IWSSD [17], BCS REFSG&FACS Joint Workshop on Requirements Engineering and Formal Methods) we learn that the interpretation of non-functionality varies considerably. Sometimes even the existence of non-functional requirements as opposed to functional ones is denied as there would be no reason for such a distinction. As a consequence, are non-functional requirements only present in the imagination of some 'crazy' persons?

In fact, most of those disagreements can be traced back to severe misunderstandings. Firstly, the points of view tend to be rather extreme - be it that some proponents of non-functional requirements state that quality needs no functionality on the one hand, and that some opponents of non-functional requirements argue that quality cannot be treated as first class citizen and therefore there is no need to make a distinction in kinds of requirements. However, it should be clear that without any function (purpose) there will be no need for quality, but just pure functionality cannot survive without quite a portion of quality neither. Hence, a forced division of the notion of requirements for mere terminological motives is far too exaggerated. Secondly, people tend to define non-functional requirements as "anything which is not obviously a functional requirement" which also can be interpreted as "anything which cannot be expressed by current ('functional') methods". But, such an artificial distinction due to our current limitations in dealing properly with some kind of requirements is by no means an adequate approach in requirements research neither.

However, it does not alter the fact that there exist requirements which cannot be expressed by mere functional specifications in the form of *input* $\Rightarrow$ *process* $\Rightarrow$ *output*. Furthermore, these requirements might even be more important than the 'functional' ones. As these requirements do not have deserved such a negatively tasted term as 'non-functional', in the sequel, it will be substituted by the more suitable term 'extra-functional' which was coined by Mary Shaw with a work on software architecture focussing on architectural properties [29].

## 3 The notion of extra-functional requirements

While functional requirements describe the intended purpose of a system component, *extra-functional* requirements (EFRs) - as seen in this paper - are constraints regarding *quality* as well as *economics* (e.g. time, cost).

Following the distinction between function and behaviour as stated in [5] and [15], a function is what is necessary or desired, and the behaviours are how this result is attained. Apart from behaviour and its function (purpose), structural information as a third view in modelling completes the picture of modelling systems. Investigating and interpreting functional requirements in a top-down manner leads to the definition of behavioural and structural models which nowadays can be represented in a quite straight-forward way using adequate modelling techniques (c.f. [6,8,14, 27,28,32,33]) which is not the case with extra-functional requirements. This and some other key differences between functional and extra-functional requirements are shown in Table 1.

| **Functional Requirements** | **Extra-functional Requirements** |
|---|---|
| describe what a system should do - *purpose* | delimit the solution space - *constraints* |
| are mainly focused by analysis process | have no fix part in analysis process |
| sufficient methods for analysis available | few methods for analysis known |
| straight-forward representation | difficult representation |
| easy to elicit | difficult to elicit |
| decomposition possible | multi-faceted structure |
| module test sufficient | often complete system necessary for verification |
| are complementary | can contradict each other |

Table 1: Functional vs extra-functional requirements

As can be seen from Table 1, the key problems originate in the nature of extra-functional requirements. As opposed to functional requirements, compliance of extra-functional requirements cannot be easily qualified to be right or wrong, the results can rather be compared along an ordinal scale giving good or bad solutions. Furthermore, the structure of extra-functional requirements can soon overcharge our capabilities in dealing with complexity. Functionalities can be structured, decomposed and represented using existing analysis techniques. Extra-functional requirements can be subject to abstraction mechanisms (like hierarchies of aggregation/decomposition and generalization/specialization), too. Additionally, they may also

- constrain special functional requirements,
- influence (positively or negatively) other EFRs,
- concern the system as a whole,
- concern the development process,
- origin from different stakeholders in the development process.

Starting from the problem of inheritance anomalies as coined in [19] in the context of object-oriented concurrent programming, it was shown in [11] that extra-functional requirements cannot be simply modelled as attachments to functional requirements. This was proved by generalizing the notion of inheritance anomalies and identifying similar problems which arise in the context of object-oriented modelling of security constraints. As long as current modelling techniques focus on functional requirements, it is useless to try to get extra-functional requirements under the same umbrella. Tight interweaving of extra-functional with functional requirements during requirements modelling can lead to major design problems resulting in complex and unreadable designs which are difficult to maintain. Thus, extensions of existing methods will not be sufficient or even lead to more interweavings of rather different contexts. Nevertheless, there exist some proposals with quite successful approaches regarding particular aspects of extra-functionality like modeling performance (c.f. [9,22,23]) or security constraints (c.f. [7,10,11]).

Above all, the kind of a requirement (be it functional or extra-functional) during the elicitation and analysis process will usually not be clear from the very beginning. Hence, a rather general goal will have to be further refined and concretized in order to result in a final classification. Additionally, it may depend on the current situation whether a requirement can be regarded as functional or as extra-functional.

However, our primary concern should not be a clear distinction between functional and extra-functional requirements irrespective of the current circumstances, but - as extra-functional requirements tend to be paid not enough attention to and therefore are easily forgotten - an adequate possibility to make all requirements explicit, be it in terms of functional or extra-functional requirements.

## 4 Dimensions of extra-functional requirements

Any attempt to comprehensively identify and classify types of extra-functional requirements will fail because of their context-sensitive nature. A fully fledged classification scheme being valid for any kind of project can never be sufficiently established. Thus, it is not surprising that there exists a variety of different classifications of non-functional requirements. There are three properties these approaches have in common: they are enumerative, not exhaustive, and rather difficult to extend. A more appropriate classification should either be exhaustive or easy to extend.

### 4.1 Generic classification scheme

Existing product-centered classification approaches propose various enumerations of software quality attributes which are sometimes ordered within a hierarchical structure (e.g. [3,13]). The actual hierarchy depends on the selected quality attributes and their relationships (as they are commonly accepted or perceived by the author). A further ordering can also take place according to different viewpoints like stakeholders primary concerns [4]. Approaches which additionally consider process and external requirements (e.g. [20,30]) also classify extra-functional requirements hierarchically and distinguish between product, process, and external requirements at the topmost level leading to three different trees.

All of the currently known classification approaches are nevertheless soundly based but lack in the ability to structure extra-functional requirements along more than one single dimension. In order to enable the incorporation and combination of different viewpoints, a multi-dimensional classification approach is required. It should be universal in the sense of being usable in the long run and in many different development processes. Adaptability to different contexts and evolving insight are calling for a flexible and extensible approach.

In the area of software reuse the faceted classification technique has already been successfully applied [25]. As shown in the sequel, this classification strategy can also be adapted for and applied to software requirements engineering. Faceted classification is a synthetic approach and enables the definition of more than just one single dimension. Classes are assembled by selecting predefined keywords from faceted lists. This approach provides higher accuracy and flexibility in classification. A faceted scheme may have several facets (dimensions) and each facet may consist of several terms.

The core facets selected in applying and adapting the faceted classification technique to extra-functional requirements are similar to those proposed in the Goal Question Metric (GQM) approach [1] which defines goals along the dimensions objects (products, processes, resources), viewpoints (actors), and issues. GQM starts with the definition of a goal which can be refined into several questions each of which can itself be refined into various metrics. GQM mainly concentrates on the refinement of (rather abstract) goals through questions leading to better quanitfyable goals.

Faceted classification does not only establish different dimensions but also semantically refines them into terms according to specific system development needs. Thus, each organization can establish its own classification scheme. The actual classes of extra-functional requirements can then be assembled by selecting the most appropriate set of terms within the given facets. The five core facets are:

- the part of the *product*, for which the requirement is a constraint
- the part of the *process*, for which the requirement is a constraint
- the *source*, which is the origin for the requirement
- the *feature*, which the requirement addresses
- the form of *representation*, which describes the requirement

The *product* and *process* facets position the requirement according to the two main dimensions recognized in software engineering and can also be found in the one-dimensional classification approaches mentioned before. The terms within these dimensions take into account that requirements will usually not only regard the software product or development process as a whole but can more precisely concern certain parts thereof.

While product and process are both target-oriented aspects of an EFR, it's origin is also quite important. Since extra-functional requirements may be ambiguous, inaccurate, and fuzzy in themselves, the *source* dimension is instrumental not only for requirements traceability but also for their further elaboration and refinement. In using the notion of 'source' instead of 'viewpoint', it can be pointed out that persons (stakeholders) will not be the only origins of requirements.

The *feature* facet allows for the possibility to express the semantic context which is addressed by the extra-functional requirement. This dimension subsumes all enumerations of previous classification approaches in an open list of EFR categories. It corresponds to the GQM dimension 'issue'. The terms within this facet will not only refer to *quality requirements* but also to *economic* (or resource) *requirements* like time and cost.

The four dimensions *process*, *product*, *source* and *feature* are regarded as core facets for the purpose of classification. Moreover, the classification framework should also tackle the problem of requirements documentation leading us to a fifth facet *representation* which describes the form of the requirement's capture. This dimension was mainly selected because of its practical impact with respect to traceability.

Table 2 shows the resulting core classification framework. It consists of the five facets together with an initial set of related terms.

| product | process | source | feature | representation |
|---|---|---|---|---|
| whole product<br>hardware<br>interface<br>whole software<br>function<br>behaviour<br>structure<br>database<br>user interface<br>documentation<br>design<br>code<br>. | procurement<br>proj. managemt.<br>analysis<br>design<br>programming<br>component test<br>system test<br>installation<br>maintenance<br>. | domain<br>general objective<br>client<br>user<br>project manager<br>law<br>economics<br>standards<br>int. guidelines<br>. | time<br>cost<br>performance<br>usability<br>availability<br>security<br>reliability<br>efficiency<br>flexibility<br>expandability<br>portability<br>maintainability<br>testability<br>understandability<br>modifyability<br>. | ER diagram<br>object diagram<br>state trans. diagr.<br>data flow diagram<br>structure chart<br>functional spec.<br>free text<br>test plan<br>. |

Table 2: Faceted classification scheme for extra-functional requirements

Additionally, a further dimension *domain* with an initially empty set of terms will help in getting together domain-specific extra-functional requirements. Thus, the terms within this dimension will strongly depend on the particular development projects.

The terms represented in Table 2 should be regarded just as examples for the purpose of illustration. An actual instance of the classification scheme will consist of far more specific terms with usually finer granularity. For example, 'time' is a rather general term which can be specialized to 'response time', 'development time', 'training period', and so on.

The origin for the presented classification scheme was the awareness of deep problems of the software engineering process. But similar issues regarding quality requirements can also be observed in the non-software areas of system development. Especially in early-phase requirements engineering a strict and rigid distinction between hardware, software, interfaces, and environment will seldom be neither wise nor feasible. The classification scheme is powerful and flexible enough to allow for a coarse initial classification favouring and documenting an integrated view of the whole system and also to support the evolution and refinement of the first analysis results into more concrete, detailed, and structured forms.

### 4.2 Impact and relevance

The main advantages of the generic classification scheme are its extensibility and the possibility to tailor its contents according to organization- and project-specific needs. Its multi-faceted nature allows for taking into account the multi-dimensional nature of extra-functional requirements leading to dynamically assembled classes. As the terms within each classification scheme instance will apply to the whole organization, a common vocabulary (concerning development phases, product and document types, EFR features) together with common development strategies will be essential byproducts.

Institutionalizing the generic classification scheme together with an adequate information system which enables the management of requirements instances by attaching values to terms and administrates also further relevant information concerning actual requirements and their compliance can substantially improve the documentation and quality of extra-functional requirements, enhance communication and the process of requirements elicitation, and support traceability and control.

*a) Requirements documentation:*
The usage of a requirements repository will succeed in better documented requirements reducing the risk of requirements "lost somewhere in the paperwork". Moreover, the structure of the classification scheme together with the repository's built-in query mechanism will help in organizing and structuring the requirements.

*b) Quality of requirements:*
As requirements will be handled in a more structured way, ambiguity can be reduced and a better degree of accuracy can be achieved. The existence of an evolving set of terms and the experience gained from former projects will contribute to a better chance of complete requirements.

*c) Process of requirements elicitation:*
The initial set of terms which will be subsequently extended by project experience can help in requirements elicitation allowing for carefully directed questions. Thus, the classification scheme can be used as an elicitation guideline. Additionally, the definition and reuse of elicitation templates in the form of generic patterns for particular EFR categories may turn out to be a valuable starting point in elicitation. As a consequence, the process of requirements elicitation will gain in effectivity as well as in efficiency.

*d) Effects on communication:*
Prospective impacts on communication are twofold. On the one hand, the interaction with the customer will become a more lively process with an analyst who is not just passively listening but is also able to play an active role by touching upon features which otherwise might have been hidden within the brain of the customer as tacit knowledge. On the other hand, the interaction to other persons in the subsequent development process is supported by stating who has to know (and act) about what requirements. Hence, the further treatment of the identified requirements can be better conducted and monitored. The latter kind of communication will take place rather in an indirect manner through querying and updating the contents of the EFR information system.

*e) Traceability:*
In using the EFR information system throughout the development process, traceability of requirements back to their origins (sources, reasons, preferences) as well as forward to their compliance (design, code, test units) can be facilitated. The attached values along the representation facet serve as pointers to the actual development documents.

*f) Control support:*
The EFR information system can also serve for control purposes. On the one hand, compliance control can be supported by respective queries and documented by updating the contents of the EFR base; on the other hand, project management can be assisted in pursuing the overall project status.

## 5. Conclusion

Mere technical methods are often insufficient and rather inadequate for dealing properly with quality requirements. The real challenges, however, consist in a commitment to the importance of quality requirements within the whole system development process. This paper provides an integrated view on requirements with special focus on extra-functional requirements (quality and economic constraints). An adaptable and extensible generic classification scheme can be used as an effective means in eliciting and representing extra-functional requirements. Nevertheless, we should be conscious of the fact that existing software engineering processes must be adapted accordingly to achieve the full integration potential.

## Acknowledgement

The author gratefully acknowledges the reviewers' comments on earlier versions this paper.

## References

[1] V.R. Basili, G. Caldiera, H.D. Rombach: "Goal Question Metric Approach", in: J.J. Marciniak (ed.): "Encyclopedia of Software", Vol. 1, 1994, pp. 528-532

[2] A. Bertino: "Guest Editor's Corner - Achieving Quality in Software", Journal on Systems and Software, Vol. 26, No. 1, July 1994, pp. 1-3

[3] B.W. Boehm: "Software Engineering", IEEE Transactions on Computers, Vol. 25, No. 12, Dec. 1976, pp. 1226-1241

[4] B.W. Boehm, H. In: "Identifying Quality Requirements Conflicts", IEEE Software, Vol. 13, No. 2, 1996, pp. 25-35

[5] B. Chandrasekaran, A.K. Goel, Y. Iwasaki: "Functional Representation as Design Rationale", IEEE Computer, Vol. 26, No. 1, Jan. 1993, pp. 48-56

[6] P. Chen: "The Entity-Relationship Model: Toward a Unified View of Data", ACM Transactions on Database Systems, Vol. 1, No. 1, 1976, pp. 9-36

[7] L. Chung: "Dealing with Security Requirements during the Development of Information Systems", Proc. CAiSE'93, Springer Verlag, Paris, 1993, pp. 234-251

[8] P. Coad, E. Yourdon: "OOA - Object-oriented Analysis", Prent. Hall, 1991

[9] A.M. Davis: "Software Requirements - Objects, Functions, and States", Prentice Hall, 2nd ed., 1993

[10] M. Dobrovnik, J. Eder: "Logical Data Independence and Modularity through Views in OODBMS", Proc. Engineering Systems Design and Analysis Conference (ESDA'96), Montpelier, 1996, pp. 13-20

[11] E. Hochmüller: "Inheritance Contradictions between Functional and Extra-functional Requirements", Proc. Second World Conference on Integrated Design & Process Technology (IDPT'96), Vol. 1, SDPS, Austin, Dec. 1996, pp. 106-113

[12] IEEE, Inc: "IEEE Guide to Software Requirements Specifications, ANSI/IEEE Std. 830-1984, 1984

[13] ISO/IEC: "International Standard ISO/IEC 9126. Information technology - Software product evaluation - Quality characteristics and guidelines for their use", Geneva, 1991

[14] M. Jackson: "System Development", Prentice Hall, 1983

[15] H. Kaindl: "An Integration of Scenarios with their Purposes in Task Modeling", Proc. ACM Symposium on Designing Interactive Systems (DIS'95), Ann Arbor, MI, August 1995
[16] S.E. Keller, L.G. Kahn, R.B. Panara: "Specifying Software Quality Requirements with Metrics", in: R.H. Thayer, M. Dorfman (eds.): "System and Software Requirements Engineering", IEEE-CSP Tutorial, 1990
[17] J. Kramer, A.L. Wolf: "Succeedings of the 8th International Workshop on Software Specification and Design", ACM Software Engineering Notes, Vol. 21, No. 5, Sep. 1996, pp. 21-35
[18] P. Loucopoulos, V. Karakostas: "System Requirements Engineering", McGraw-Hill, 1995
[19] S. Matsuoka, A. Yonezawa: "Analysis of Inheritance Anomaly in Object-Oriented Concurrent Programming Languages", in: G. Agha, P. Wegner, A. Yonezawa (eds.): Research Directions in Concurrent Object-Oriented Programming, MIT Press, 1993, pp. 107-150
[20] R.T. Mittermeir: "Requirements Engineering", in: K. Kurbel, H. Strunz: "Handbuch Wirtschaftsinformatik", Poeschel Verlag, 1990
[21] R.T. Mittermeir: "Dimensions of Software Design - From Algorithms to Systems", Proc. IDPT'96, Vol. 1, SDPS, Austin, Dec. 1996, pp. 82-89
[22] B.A. Nixon: "Dealing with Performance Requirements During the Development of Information Systems", Proc. IEEE International Symposium On Requirements Engineering (RE'93), San Diego, 1993 pp. 42-49
[23] A. Opdahl: "Requirements Engineering for Software Performance", Proc. REFSQ'94, Utrecht, 1994, pp. 16-32
[24] C. Potts: "Requirements Models in Context", Proc. Third IEEE International Symposium On Requirements Engineering (RE'97), Annapolis, 1997, pp. 102-104
[25] R. Prieto-Diaz: "Implementing Faceted Classification for Software Reuse", ACM Communications, Vol. 34, No. 5, May 1991, pp. 88-97
[26] G.-C. Roman: "A taxonomy of current issues in requirements engineering", IEEE Computer, Vol. 18, No. 4, April 1985, pp.14-23
[27] D.T. Ross, K.E. Schoman: "Structured Analysis for Requirements Definition", IEEE Transactions on Software Engineering, Vol. 3, No. 1, 1977, pp. 1-65
[28] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen: "Object-oriented Modelling and Design", Prentice Hall, 1991
[29] M. Shaw: "Truth vs Knowledge: The Difference Between What a Component Does and What We Know It Does", Proc. 8th IWSSD, Schloss Velen, Germany, March 1996, pp. 181-185
[30] I. Sommerville: "Software Engineering", Addison-Wesley, 1992
[31] G. Starke: "Session Summary: Non-Functional Requirements", Proc. REFSQ'94, 1994, pp. 4-6
[32] E. Yourdon, L.L. Constantine: "Structured Design", Prentice Hall, 1979
[33] E. Yourdon: "Modern Structured Analysis", Prentice Hall, 1989