QUALITY IMPROVEMENT THROUGH QUALITY REQUIREMENTS MANAGEMENT

Elke Hochmüller

Institut für Informatik
Universität Klagenfurt

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 emphasizes quality requirements and their integration into the system development process. Based on a faceted and adaptable classification approach, a framework to manage quality requirements will be proposed. Prospective effects improving not only the quality of the elicited requirements and their compliances but also the quality of the software development process itself will be discussed.

Keywords: extra-functional requirements (EFRs), faceted requirements classification, requirements management.

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 (cf. [15, 16]). 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 [13]. 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, non-behavioral requirements, (software) quality requirements, extra-functional requirements or simply '-ilities' (cf. [2, 6, 9, 10, 11, 18, 21]).

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 [1]. One of the main objectives of the approach presented in the sequel is to overcome these problems by explicitly focussing on quality requirements throughout the overall system development project.

The intention of this paper is not only to provide a clear starting point to characterize and manage 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 [9]. In fact, 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. In the sequel, these requirements will be referred to as 'extra-functional requirements' (EFRs) subsuming not only *quality requirements* (constraints regarding security, usability, maintainability, ...) but also *economic* (or *resource*) *requirements* (like time and cost constraints).

The key problems of extra-functional requirements are mainly based on their soft and sometimes even fuzzy nature. They are difficult to elicit and to represent. Hence, EFRs are neglected by all the well-known analysis and modelling techniques. 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. However, they may additionally

- 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 by [14] in the context of object-oriented concurrent programming, it was shown in [8] 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.

It should become obvious from this discussion that extra-functional requirements are critical success factors in software development. Hence, they should be regarded as valuable assets which deserve much more attention during software development projects. A better and organized management of extra-functional requirements could substantially contribute to improve the process of requirements elicitation leading to more accurate and complete requirements which can be traced back to their origins as well as forward to their compliances. This would lead to extra-functional requirements which are recognized as integral parts of the software development process. For the purpose of integration, the generic framework proposed in [9] which classifies extra-functional requirements along a multi-dimensional scheme can be applied in order to capture EFRs and subsequently administrate them throughout the whole software development life cycle.

The rest of the paper is organized as follows. Section 2 contains an overview of the generic classification framework. Practical considerations regarding the management of extra-functional requirements on basis of the classification framework will be discussed in section 3. Section 4 deals with prerequisites for an adequate EFR management. Potential improvements on the quality of the requirements and parts of the development process will be outlined in section 5 which will be followed by some concluding remarks.

2. Generic Classification Framework

A universal characterization scheme of extra-functional requirements must be 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 [17]. 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. 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 and each facet may have several terms.

Requirements engineering is obviously a process (involving stakeholders) of establishing a (composite) product. The different stakeholders within this process can be regarded as sources of requirements on certain features of parts of the product or/and of parts of the development process. These requirements are captures in terms of representations. Hence, the following five facets are proposed as core dimensions:

- the part of the *product*, for which the requirement is a constraint
- the part of the *process*, for which the requirement is a constraint

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

Table 1: Faceted classification scheme for extra-functional requirements

- the *source*, which is the requirement's origin
- the *feature*, which the requirement addresses
- the form of *representation*, which describes the requirement

The more general dimension *target* subsuming the *product* as well as the *process* dimension was decided to split up in favor of being able to express the possibility of attaching a requirement to both a part of a product and a part of a process.

Table 1 shows the core classification framework as described in [9]. It consists of the above five facets together with an initial set of related terms. The terms within each facet can be defined according to the 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.

An additional 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.

It is noteworthy that the dimension *feature* subsumes all results of previous single-dimensional classification approaches in an open list of EFR categories. These terms refer to *quality requirements* as well as to *economic* (or resource) *requirements*.

The terms represented in Table 1 should be considered 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 favoring 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.

3. Extra-Functional Requirements Management

The real benefits of the generic classification scheme can only be exploited if it is used as basis for a proper *requirements management* of extra-functional requirements.

Limiting the focus of requirements management to EFRs should not be regarded as a constraint but it is rather feasible to restrict the additional management overhead to an inevitable minimum. The following suggestions can be also valid for functional requirements, but these usually can be well expressed by existing requirements modelling techniques and their realization and compliance test can be carried out in a quite straight-forward manner. Hence, there should be no need for any extra administration effort regarding functional requirements.

Extra-functional requirements, on the other hand, cannot be properly treated using existing modelling techniques (cf. [4, 5, 12, 19, 20, 22, 23]) which are mainly focussing on static and dynamic system properties and almost totally neglecting resource as well as quality requirements. Extensions to these methods can only be effective for some selected types of extra-functional requirements (e.g. performance, security constraints) but not at all be sophisticated enough to tackle all possible kinds of EFRs. Additionally, such enhanced methods will still have to prove their true applicability, as - because of interweavings of extra-functional with functional requirements - the resulting model components might turn out to be rather cumbersome regarding their complexity raising problems with respect to readability and maintainability [8].

This leaves us with a situation which can be described as follows: Functional requirements can be captured, expressed, and realized quite well in applying well-known and effective analysis, design and programming techniques. Existing methods, however, are rather inadequate for analyzing and expressing extra-functional requirements. The lack in being able to cope with EFRs in a structured way can lead to misunderstood, forgotten and neglected requirements which might be the real reasons for most of the doomed projects.

The absence of powerful technical methods for dealing with extra-functional requirements calls for a compensating strategy in making EFRs explicit. One approach which is currently feasible and can be realized within a short time is the use of pure organizational means.

Using the generic classification scheme as meta information for EFR categories, we can establish an EFR information system which supports the treatment of extra-functional requirements throughout the system development process.

For this purpose, the definition of instances of each synthesized class must be possible. This can be achieved by associating values to each selected term resulting in combinations of triples of the common form *(facet, term, value)*.

Anchoring the actual requirement within the classification framework enables not only its characterization but provides also an initial set of information about the target, the source, the feature, and the representation of that requirement. However, in order to take full advantage of the potential of such an information system, additional information about captures and relationships of the actual requirement as well as about its compliance and subsequent compliance control is necessary to be administered.

a) Requirements capture information:
Further essential information deals with the viability of the requirement and the circumstances of its capture. Hence, the following items are proposed as a minimum set of requirements capture information:

- preference or degree of importance
- degree of stability
- reason for actual requirement
- identification of analyst
- date of entry

b) Requirements relationship information:
Especially EFRs can not be considered in isolation. They often will concern (constrain) other functional requirements. These relationships are already covered by the usage of the classification schema itself (as the values within the dimensions *product* and *domain* will usually refer to functional requirements). However, there will be different kinds of interrelations between EFRs themselves (e.g. hierarchies of aggregation and generalization, positive and negative dependency relationships). Requirements can also evolve and be subject to changes which will require to keep track of that evolution. Thus, the administration of the following relationships seems advisable:

- aggregation/decomposition hierarchies
- generalization/specialization hierarchies
- (kinds of) relationships to other EFRs
- versioning information

c) Compliance plan information:
The usage of an EFR information system should not be confined to expressing and storing requirements but also help in planning their compliance as well as the subsequent control of their compliance. Such planning information can be:

- date/phase of earliest possibility for compliance
- date/phase of latest possibility for compliance
- compliance control plan (method of compliance control)
- reaction in case of failed compliance control

d) Compliance information:
The administration of the following information about the actual compliances can help in gaining a continuous survey of the current project status:

- reference to compliance product
- date of actual compliance
- identification of person in charge for compliance

e) Compliance control information:
The EFR information system will contain valuable information regarding the overall project. This information can be used for compliance control as an essential part of quality assurance activities. Results of such reviews can again be entered into the information system in order to support control documentation. These inputs can include:

- result of compliance control
- date of compliance control
- identification of person in charge for compliance control

The EFR information system will hold a possibly huge volume of data with a lot of cross-references between the items. Powerful mechanisms for the search of items and the navigation between them are essential for the usefulness of the system. This makes it advisable to use a database system in combination with hypertext technology as the underlying framework for the EFR information system.

4. Implications for the Development Process

Although extra-functional requirements are critical success factors, they are too often neglected in practice. It should be our aim to fortify their relevance within the system development process from the very beginning. For this purpose, a full integration of extra-functional requirements into the system development process as well as into the organizational structure should be striven for. This integration should take place as smooth as possible by fully exploiting currently applied and approved development strategies which will be enriched by EFR specific activities based on an adequate infrastructure.

4.1. EFR Infrastructure

From the organizational point of view, proper structures are required to support EFR management.

a) EFR Meta Schema
The main advantages of the generic classification schema are its extensibility and the possibility to tailor its contents according to organization- and project-specific needs. Thus, each organization can establish its own classification schema instance which in its turn will serve as a meta schema for the actual EFR information base containing EFR-related information.

b) EFR Information Base
EFR management will only take place effectively and efficiently if the activities are supported by an adequate information system. The so-called EFR information base will keep track of all the information as described in the previous section.

c) EFR Elicitation Guidelines and Templates
Once institutionalized, the EFR classification schema together with domain-specific experience in requirements elicitation and analysis can help in advanced EFR management. This means that guidelines for future EFR elicitation can evolve through practice and continuous use of the EFR classification framework. The definition and reuse of elicitation templates in the form of generic patterns for particular EFR categories can be of essential help during elicitation.

4.2. EFR Advocate

The establishment of a additional role in terms of a project independent *EFR advocate* (or *EFR advocate group* with a number of members appropriate to the size of the company) will help to pay adequate and continuous attention to extra-functional requirements.

The main concern of the EFR advocate should be to act in behalf of extra-functional requirements. Related activities include the administration of the EFR infrastructure, the responsibility of EFR specific technology transfer as well as the accomplishment of EFR-related tasks during different stages of the actual system development processes.

a) EFR infrastructure administration
The EFR advocate will be responsible for the administration and maintenance of the meta information constituting the generic classification scheme and the contents of the EFR information base, and - as experience grows - the structure of the EFR templates.

b) EFR specific technology transfer
For members of project groups the EFR advocate should also act as an interlocutor regarding EFR concerns (with special focus on EFR-related elicitation education).

c) Requirements analysis
The EFR advocate should be present and active during requirements engineering and act as an analyst with particular sensitivity regarding extra-functional requirements. Related activities include the identification, refinement (ask for further information in case of rather high level goals or objectives which are difficult to quantify), classification, and documentation (together with insertion into the EFR information base) of extra-functional requirements.

d) Review and Control
The EFR advocate should also be in charge of EFR specific reviews and the final internal compliance control.

Do we really need a new role just for extra-functional requirements? Well, if we take into account all those horrible stories regarding failed project because of poor quality products - isn't it worth to spend some overhead in favor of acceptable quality and therefore successful projects? The break-even-point should rather soon be attained if we regard the losses which otherwise could occur. Recent discussions on the relevance of SQA (software quality assurance) groups (cf. [7]) center on the criticism of their role as an 'end-item' checker. It is quite conceivable that the SQA group could take on the EFR-related activities described above to become a more effective advocate in favor of quality.

Another organizational approach for achieving higher quality proposes the formation of a Requirements Identification and Formulation Team (RIFT) which should be established within a customer organization [3]. This should enable customer organizations themselves to contribute to the reduction in the failure rate of system development project. However, most of the suggested guidelines for establishing a RIFT do not consider time, personnel capacity, and budget aspects. Additionally, multiple education effort has to be taken into consideration when establishing a RIFT in each customer organization.

The role of an EFR advocate, on the other hand, should be installed within the developer organization. Hence, the skills of the EFR advocate can be utilized for many development projects. In such a way, experience in EFR management has a higher potential to grow with each additional project.

5. Potential for Quality Improvement

The potential benefit of the described EFR-specific effort regarding the organizational structure as well as the development process will concern the quality of the requirements themselves as well as the quality of the system development process which in turn will have positive effects on the quality of the resulting system.

5.1. Requirements Quality

Institutionalizing an organization-specific instance of the generic classification schema together with an adequate EFR information base which enables the management of actual requirements and their compliances will lead to higher requirements quality.

Using the organization-wide EFR infrastructure calls for a structured way of handling extra-functional requirements which will reduce ambiguity and lead to a better degree of accuracy in requirements formulation. The existence of an evolving set of terms and the experience gained from former projects will contribute to a better chance of complete requirements. The usage of the EFR information base will improve requirements documentation reducing the risk of EFRs "lost somewhere in the paperwork". Using the EFR information base throughout the development process allows for traced requirements (references to origins in terms of sources, reasons, preferences) as well as traceable requirements (cross-references between requirements and their compliances in terms of design, code, or test units) in the sense of [6].

5.2. Process Quality

EFR management activities which are fully integrated into the software development process will contribute to different facets of improved process quality.

a) Customer Interaction
The interaction with the customer will be a more active process with an EFR-minded analyst who will not just listening but touch upon certain EFR features which otherwise might stay hidden within the brain of the customer as tacit knowledge.

b) Improved Requirements Elicitation
Experience with EFRs as well as with EFR-related infrastructure will certainly improve the process of requirements elicitation. The initial set of terms within the classification framework which will be continuously extended by project experience allows for carefully directed questions. Templates for particular EFR features can be a valuable starting point for elicitation. Thus, the process of requirements elicitation will gain in effectivity and efficiency.

c) Conflict identification and negotiation
An important advantage of systematic EFR management is the increased transparency which can be achieved by EFR documentation by means of the EFR information base. This transparency provides the essential inputs for the identification of conflicting EFRs and contributes the necessary information to an early and effective process of negotiation.

d) Downstream Communication
The usage of the EFR information base throughout the *requirement life cycle* (identification, compliance plan, compliance, compliance control) supports the so-called downstream communication from the analysts to the designers, programmers, testers. The EFR information base contains information about who has to know and act about what requirements. Thus, further treatment of the identified requirements can be better conducted and monitored.

e) Organized Compliance and Project Control
Continuous usage of the EFR base during the whole software development project will also support the processes of compliance control (control in the small) and project control (control in the large). During compliance control the actual software components will be examined regarding the degree of EFR satisfaction. The information necessary for this purpose can easily be obtained and documented from/into the EFR information base. Project control deals with the observation of the overall project status. A rough overview of a project can be easily obtained from the EFR information base by respective aggregating queries.

f) Internal Standardization
As the terms within each classification schema 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.

6. Concluding Remarks

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 for system development. This paper focusses on extra-functional requirements (quality and economic constraints) and their integration into the system development process by means of a proper EFR management. A generic classification framework can serve as basic instrument for the elicitation and documentation of EFRs. In order to achieve the full integration potential, however, existing software engineering processes must be adapted accordingly.

References

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

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

[3] D. Brash, B. Wangler: "Bridging the RIFT Between Customers and Developers", Proc. Third International Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ'97), Barcelona, June 1997

[4] 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

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

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

[7] J. Henry, B. Blasewitz: "Do we really need SQA to produce quality software? No! Well maybe. It depends. Yes!", ACM SIGSOFT Software Engineering Notes, Vol. 19, No. 2, April 1994, pp. 63-64

[8] 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

[9] E. Hochmüller: "Requirements Classification as a First Step to Grasp Quality Requirements", Proc. Third International Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ'97), Barcelona, June 1997

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

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

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

[13] P. Loucopoulos, V. Karakostas: "System Requirements Engineering", McGraw-Hill, 1995

[14] 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

[15] R.T. Mittermeir: "Dimensions of Software Design - From Algorithms to Systems", Proc. Second World Conference on Integrated Design & Process Technology (IDPT'96), Vol. 1, SDPS, Austin, Dec. 1996, pp. 82-89

[16] C. Potts: "Fitness for Use: The System Quality that Matters Most", Proc. Third International Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ'97), Barcelona, June 1997

[17] R. Prieto-Diaz: "Implementing Faceted Classification for Software Reuse", ACM Communications, Vol. 34, No. 5, May 1991, pp. 88-97

[18] G.-C. Roman: "A taxonomy of current issues in requirements engineering", IEEE Computer, Vol. 18, No. 4, April 1985, pp.14-23

[19] D.T. Ross, K.E. Schoman: "Structured Analysis for Requirements Definition", IEEE Transactions on Software Engineering, Vol. 3, No. 1, 1977, pp. 1-65

[20] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen: "Object-oriented Modelling and Design", Prentice Hall, 1991

[21] G. Starke: "Session Summary: Non-Functional Requirements", Proc. First International Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ'94), 1994, pp. 4-6

[22] E. Yourdon, L.L. Constantine: "Structured Design", Prentice Hall, 1979

[23] E. Yourdon: "Modern Structured Analysis", Prentice Hall, 1989