TOWARDS THE PROPER INTEGRATION OF EXTRA-FUNCTIONAL REQUIREMENTS

Elke Hochmüller

Department of Telematics/Network Engineering
University for Applied Sciences
Carinthia Tech Institute / Technikum Kärnten
Primoschgasse 3, A-9020 Klagenfurt
AUSTRIA
hochm@cti.ac.at

ABSTRACT

In spite of the many achievements in software engineering, proper treatment of extra-functional requirements (also known as non-functional requirements) within the software development process is still a challenge to our discipline. The application of functionality-biased software development methodologies can lead to major contradictions in the joint modelling of functional and extra-functional requirements.
Based on a thorough discussion on the nature of extra-functional requirements as well as on open issues in coping with them, this paper emphasizes the role of extra-functional requirements in the software development process. Particularly, a framework supporting the explicit integration of extra-functional requirements into a conventional phase-driven process model is proposed and outlined.

Keywords: extra-functional requirements, process modelling, integration framework

MOTIVATION

Many advances have been achieved in the area of software engineering during the last 30 years. The main breakthrough was certainly the shift from the highly artistic way of programming to a systematic and phase-structured process of software development. Many process models were proposed and applied so far. In order to support the various activities within the software development process, a broad spectrum of methods, techniques, and tools is currently available. Most of them concentrate on functionality. But functionality is not the only dimension that matters in software development (cf. Mittermeir 1996; Potts 1997(a)). A whole bunch of additional software product dimensions is put up by product quality requirements (like security, performance, usability). These requirements may be at least as important as the functional ones since a product must possess an acceptable degree of quality such that it becomes worth being used (cf. fitness for use: Potts 1997(b)). In the sequel, these extra categories of requirements will be simply called extra-functional requirements.
The deficiencies of giving extra-functional 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. Currently available and applied (analysis and design) methods, techniques and tools mainly focus on objects, functions and states. 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 obviously quality cannot be added to a software product a-posteriori (Bertolino 1994). Moreover, quality has to be ensured from the very beginning of the software life cycle (Rai et al. 1998). This requires to inherently incorporate quality related aspects into each software development process.
This paper serves four purposes. First, it shows that there definitely are extra-functional requirements which cannot be treated as mere attachments to functions (cf. chapter 2). Second, the process-related challenges in dealing with extra-functional requirements are elaborated (cf. chapter 3). Third, the role of extra-functional requirements within the software development process as well as within the software product is discussed (cf. chapter 4). Fourth, a framework is proposed for a better integration of extra-functional requirements with existing phase-oriented process models (cf. chapter 5).

EXTRA-FUNCTIONAL REQUIREMENTS

While functional requirements describe the intended purpose of a system component, extra-functional requirements (EFRs) are constraints regarding *quality* (e.g. performance, maintainability, usability) as well as *economics* (e.g. time, cost) of process or product components. Distinguishing between function and behavior (cf. Chandrasekaran et al. 1993; Kaindl 1995), a function is what is necessary or desired (e.g. a car should be able to transport persons) and the behaviors are how this result is attained (e.g. a car should run on streets). In our car example, extra-functional requirements may be stated as environmental conditions like:

- The car need not run on water but on sealed roads.
- With a lower speed it should also be able to run on bumpy roads.

Similar conditions may also matter in the case of software systems. Table 1 contains some examples which illustrate the analogy between extra-functional requirements of 'real-life' systems (e.g. car) and software systems (e.g. distributed flight reservation system).

Car	Flight Reservation System (FRS)
The car should run at least 8000 miles without oil change.	The FRS should be available 99.99 percent of the time in any given calendar year.
Break, clutch and accelerator pedals should be arranged according their standard positions.	Standard elements should be used for the user interface (e.g. windows, icons, mouse).
Indicating instruments should be independent of the driver's nationality (e.g. usage of icons).	The user interface should be multilingual (e.g. English and German; icons).
Experienced drivers should be able to use the car within 10 minutes of preparation.	Experienced computer users should be able to use the system after a training period of 1 hour.
The car should still operate in case of 20 percent of overweight.	If the initial amount of users increases by 50 percent, the response time must not increase by more than 20 percent.
The car should dispose of two independent break mechanisms.	In case of a network failure, the local system should be able to operate independently.
Car driving should not keep the driver fully occupied such that not even a little chat with the passengers is possible.	The usage of the FRS should not hinder from talking with clients in the office or via the phone.
A service man should be able to check 4 cars per hour.	The technician should be able to carry out his maintaining tasks on the whole system within his 20 office hours.
The car should satisfy legal provisions (e.g. exhaust gas limits).	The FRS must be certified concerning the specified security system.

Table 1. Extra-functional requirements - examples

Analysing functional requirements in a top-down manner leads to the definition of behavioral and structural models which nowadays can be represented in a quite straight-forward way using adequate modelling techniques, e.g. SA/SD (Yourdon 1978; Yourdon 1989), ER (Chen 1976), JSD (Jackson 1983), OOA (Coad & Yourdon 1991), OMT (Rumbaugh et al. 1991), SADT (Ross & Schoman 1977), UML (Rumbaugh et al. 1998). As opposed to functional requirements, extra-functional requirements are not embodied thoroughly in the analysis process. They are difficult to elicit and to represent, have a multi-faceted structure, may constrain special functional requirements, the system as a whole or the development process on its own, influence (positively or negatively) other extra-functional requirements. Compliance of extra-functional requirements cannot easily be qualified as right or wrong, the results can rather be compared along an ordinal scale giving good or bad solutions (Loucopoulos & Karakostas 1995). The structure of extra-functional requirements can soon overcharge our capabilities in dealing with complexity. Table 2 illustrates the contrast between functional and extra-functional requirements.

Functional Requirements	Extra-functional Requirements
describe what a system should do - *purpose*	delimit the solution space – *constraints*
are mainly focused by analysis process	are not inherently embodied in analysis process
sufficient methods for analysis available	few methods for analysis known
standard techniques/notations for representation	no representation standards
easy to elicit	difficult to elicit
decomposition possible	multi-faceted structure
module test sufficient	often complete system necessary for verification
are largely independent of each other	can contradict each other

Table 2. Functional vs extra-functional requirements

Extra-functional Requirements are Different

Within the computer science community, the interpretation of extra-functionality varies considerably. Symptomatic for this situation are the many different definitions in the literature (cf. section 2.2.) and recent debates (REFSQ'94 (Starke 1994), IWSSD-8 (Kramer & Wolf 1996), BCS REFSG&FACS Joint Workshop on Requirements Engineering and Formal Methods, REFSQ'97 (Dubois et al. 1997)) about the existence of any independent extra-functional dimension.
Certainly, there are some kinds of extra-functional requirements which can simply be attached to functional requirements. For example, performance requirements can be quantified in terms of time periods; existing modelling techniques like state transition diagrams or data flow diagrams can easily be extended to express such time constraints (cf. Davis 1993).
Other kinds of extra-functional requirements, however, cannot be simply modelled as attachments to functional requirements. In the sequel, this will be illustrated by examples from the treatment of synchronization constraints in object-oriented concurrent programming (OOCP) and security constraints in OODBMSs representing two very different categories of extra-functional requirements originating from two different fields of computer science.

Inheritance Contradictions because of Synchronization Constraints in OOCP

In object-oriented concurrent programming, messages are allowed to arrive at an object in any order and even simultaneously. For the purpose of maintaining its internal integrity, an object in a certain state may only accept a subset of its entire set of messages. Such a restriction on the acceptable message set is called *synchronization constraint*. The object behavior that satisfies a synchronization constraint is implemented by a particular portion of the method code which is referred to as *synchronization code*. The fact that synchronization code cannot be effectively inherited without non-trivial class re-definitions was identified as *inheritance anomaly* and is deeply discussed by Matsuoka & Yonezawa (1993) using the example of a bounded buffer to illustrate the three different categories of anomalies:

- state partitioning anomaly
- history-only sensitive anomaly
- state modification anomaly

Using BoundedBuffer as common superclass, the OMT object schema in Fig. 1 outlines an example for each of the three different anomalies arising on subclass definition.

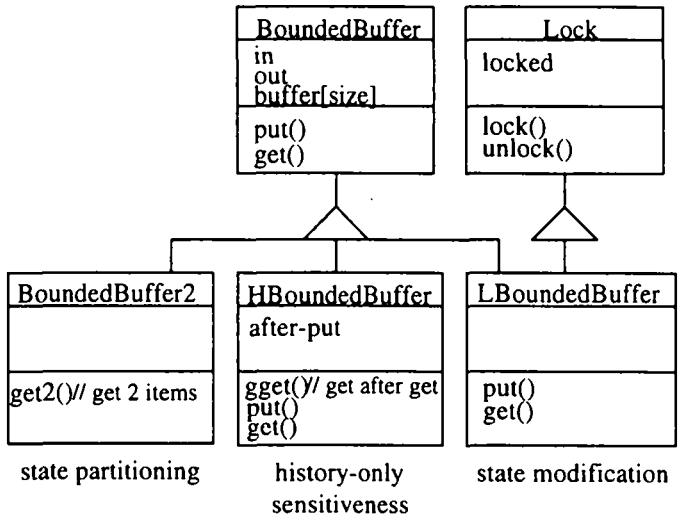

Fig. 1. Inheritance anomalies in OOCP

Subclass BoundedBuffer2 inherits the methods put() and get() from BoundedBuffer and additionally defines a method get2() which automatically removes two elements from the buffer. The associated synchronization constraint allows instances of BoundedBuffer2 only to accept message get2() if the buffer contains at least two elements. Hence, the state allowing the acceptance of both get() and put() has to be partitioned into two states in order to be able to distinguish the case where just one element is in the buffer and get2() is not allowed. This is a simple example for a state partitioning anomaly. While state partitioning anomalies can be overcome by using

method guards as synchronization mechanism, the next two types of inheritance anomalies are much more problematic to deal with.
Subclass HBoundedBuffer defines a new method gget() which behaves like method get() inherited from BoundedBuffer but cannot be accepted immediately after the invocation of put(). A new instance variable after-put for recording the necessary history information has to be defined. As after-put has to be adjusted in the methods get() and put(), the synchronization code of both methods have to be overridden in subclass HBoundedBuffer and history-only sensitive anomaly arises.
State modification anomalies can arise in the presence of multiple inheritance. Let us consider subclass LBoundedBuffer which additionally inherits the instance variable locked from a second superclass Lock. The inherited method lock() orthogonally restricts the set of states in which each method inherited from BoundedBuffer can be invoked. In order to be able to evaluate the instance variable locked, the synchronization parts of methods get() and put() must be overridden in subclass LBoundedBuffer.
Classes HBoundedBuffer and LBoundedBuffer are expressive examples for method overriding which becomes necessary in order to guarantee compliance of the corresponding synchronization constraints. However, the pure functionality of get() and put() remains the same (get() removes one element from the buffer; put() adds one element to the buffer). The original work (Matsuoka & Yonezawa 1993) on the problem of inheritance anomaly did not explicitly address extra-functional requirements. Nevertheless, it is a proof of the existence of a particular additional dimension which is something else than just pure functionality.
Although the used example is very simple, it did already demonstrate that tight interweaved treatment of functional requirements and extra-functional requirements (like synchronization constraints) can lead to major development problems resulting in rather complex and unreadable software components which are difficult to maintain.

Schema Distortion through Security Constraints in OODBMSs

Similar anomalies can arise in the context of modelling security for object-oriented database management systems (OODBMSs). Especially mandatory security models which represent multi-level real-world entities as single-level objects may have grave impacts on conceptual schema design. This can be illustrated even by such a small example as shown in Fig. 2.
Mandatory access control requires to label all *objects* in a database based on their sensitivity (*classification*). The active processes requesting access to database objects are called *subjects* and are assigned to a *clearance* level. A strictly enforced central security policy (Bell-LaPadula concept) disallows the information flow from (data) objects classified for a higher security level to subjects with a lower security clearance. Most mandatory security models for OODBMSs (e.g. message filter model described by Jajodia et al., 1995) require each object class to be classified at one single level such that all attributes and methods of an object have the same classification.
Fig. 2 represents the realization of a simple security constraint. Let us assume type Employee and subtype Manager to be part of an OMT object schema. Type elements can be either unclassified (U) or secure (S). Employee has unclassified attributes ssn and name and a secure attribute salary. Type Manager has an additional unclassified attribute bonus. Most multilevel approaches split each of the conceptual types Employee and Manager into two separate single-level parts, a secure and an unclassified one. In Fig. 2, the unfamiliar dashed lines linking the secure and the unclassified components of each of the two conceptual types may be interpreted either as security induced inheritance (Jajodia et al. 1995) or as association between composite objects (Bertino & Jajodia 1993). Both variants provide means to view just the unclassified parts or to access the whole conceptual object.

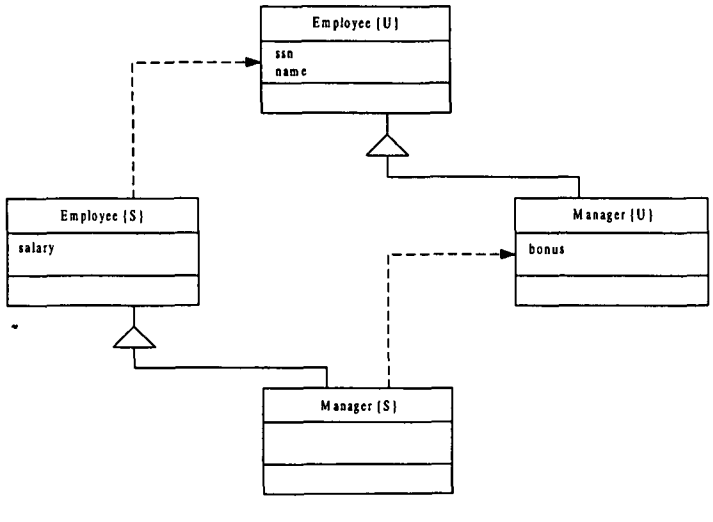

Fig. 2. Simple security constraint and multilevel classes

In the presence of more classification levels and/or more complex security constraints (cf. Dobrovnik & Hochmüller 1998) the single-level implementation strategy can lead to rather complicated structures in the conceptual schema. The pure conceptual semantics of the schema are heavily distorted by the realization of security requirements. The following anomalies resulting from tight interweaving of semantic (conceptual) and security dimensions were identified and discussed elsewhere (Hochmüller 1998):

- class partitioning
- class rearrangement
- object reassignment

How to avoid the contradictions between both dimensions by using object-oriented views as security mechanism was already outlined previously in Hochmüller 1998 and thoroughly demonstrated by Dobrovnik & Hochmüller (1998).
An in-depth discussion of limits in inheritance structures in object-oriented system development (synchronization in OOCP and security in OODBMSs) arising from tight interweavings of functional and extra-functional requirements - which cannot be simply overcome by multiple inheritance - can be found in Hochmüller 1998.
Other kinds of extra-functional requirements like the often cited cost and performance criteria are less problematic because they can usually be attached to specific functions quite easily. The examples from the security and synchronization areas given above are qualitatively different since no function can be identified to attach them naturally to. Thus, there are definitely extra-functional requirements which cannot be treated as add-ons.

Taxonomy

Previous work in the area of extra-functional requirements was mainly concerned with their classification. In the literature, various synonyms for extra-functional requirements (EFRs) are used, e.g. non-functional requirements (NFRs), non-behavioral requirements, or (software) quality requirements. Although the term 'non-functional' is used more often than the others, it has a far too negative touch. Also, it is misleading in the sense that it suggests the existence of requirements which have nothing to do with functionality, at all. However, non-functional requirements will sooner or later be incorporated in functional solutions. The debates within the scientific community about the (non-)existence of non-functional requirements mainly originate from that misinterpretation. Hence, the term 'extra-functional' which originally was coined by Mary Shaw in the context of software architectures (Shaw 1996) seems to be more appropriate to denote something which does not directly address the functionality/purpose of a system and which may additionally impose constraints on the functionality dimension. Consequently, throughout the sequel, the term 'extra-functional' will be used systematically to avoid the possible irritation due to the term 'non-functional'. However, since the semantics of both terms do not differ, they could also be used as synonyms, if the reader prefers to do so.
One of the first characterization approaches dates back to 1976 when Boehm introduced a hierarchical list of quality attributes (Boehm 1976). His recent work (Boehm & In 1996) includes the mapping of primary concerns of stakeholders onto these quality criteria in order to get a better starting point for negotiations in case of conflicting requirements. Davis (1993) prefers the notions of behavioral and non-behavioral requirements and focusses primarily on reliability and efficiency issues. The IEEE Std. 830 (1993) defines non-functional requirements in terms of performance, external interfaces, design constraints, and quality attributes. In addition to functionality, the ISO 9126 Standard (1991) distinguishes five classes of non-functional requirements as main characteristics for evaluation of product quality, namely reliability, usability, efficiency, maintainability, and portability. Roman (1985) classifies non-functional requirements according to interface constraints, performance constraints, operating constraints, life cycle constraints, economic constraints, and political constraints. A methodology (Keller et al. 1990) 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).
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 of Mittermeir (1990) and of Sommerville (1992) distinguishing between product requirements, process requirements, and external requirements, which can be legal or economic

constraints placed on both product and process. Dobson & McDermid (1991) try to avoid an explicit distinction between extra-functional and functional requirements and propose 4 kinds of (product as well as process) concepts (commitments to specific policies, obligations which are functionality-specific constraints, external constraints, and derived properties). Here, classical extra-functional requirements can be mainly perceived as obligations as they are specific goals for particular systems but may also be assigned to commitments because of their generality.

As can be seen from the amount of and differences between all of those classification approaches, there does not exist one single taxonomy which is powerful enough to identify and classify comprehensively all types of extra-functional requirements regardless of the prevailing context. Because of the multi-faceted and context-dependent nature of extra-functional requirements, chances are low for establishing a fully fledged classification schema which is suitable for any kind of project.

Existing product-centered classification approaches propose various enumerations of software quality attributes which are sometimes ordered within a hierarchical structure (e.g. Boehm 1976; ISO/IEC 1991). 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 primary concerns of stakeholders (Boehm & In 1996). Approaches which additionally consider process and external requirements (e.g. Mittermeir 1990; Sommerville 1992) also classify extra-functional requirements hierarchically and distinguish between product, process, and external requirements at the topmost level leading to three different trees.

All these classification approaches have certainly their eligibilities, but lack in the ability to structure extra-functional requirements along more than one single dimension. The generic classification framework for extra-functional requirements presented in Hochmüller 1997(a) is based on the faceted classification technique which has already been successfully applied in the area of software reuse (Prieto-Diaz 1991). 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 schema may have several facets (dimensions) and each facet may consist of several terms.

The core dimensions of the faceted classification schema can be roughly summarized when regarding the different ingredients of requirements engineering. It is obvious, that requirements engineering is a process to establish a (composite) product. Different stakeholders (with different viewpoints) are involved in the requirements engineering process. They can be regarded as sources of requirements on certain features of parts of the product or/and of parts (phases) of the development process. These requirements are captured 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 (e.g. functions and procedures, behavioral and structural components, user interface)
- the phase of the *process*, for which the requirement is a constraint (e.g. analysis, design, or programming phases)
- the *source*, which is the origin for the requirement (e.g. different stakeholders, law, standards)
- the *feature*, which the requirement addresses (actual EFR category)
- the form of *representation*, which describes the requirement (e.g. just free text or annotations in entity-relationship diagrams, state transition diagrams, ...)

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

Table 3. Faceted classification schema for extra-functional requirements

Table 3 shows the resulting core classification framework. It consists of the five facets together with an initial set of related terms. These terms 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; similarly, the term 'cost' can be further refined and also attached to product as well as process components.
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 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 (Basili et al. 1994) 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 quantifyable 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 schema. The actual classes of extra-functional requirements can then be assembled by selecting the most appropriate set of terms within the given facets.
Its extensibility and the possibility to tailor its contents according to organization and project-specific needs are also the main advantages of the generic classification schema. 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 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 valuable byproducts.

PROCESS CHALLENGES

In order to aim at acceptable approaches in dealing with extra-functional requirements, a better understanding of the challenges to be addressed during the software development process is necessary. For this purpose, let us first identify the problems which become evident when extra-functional requirements are involved in the software development process.

- ***Vagueness.*** Especially in early-phase requirements engineering, extra-functional requirements tend to be fuzzy in their nature. They may be difficult to quantify and may lack in further details (e.g. product or process component to be addressed, reason, volatility). Hence, it is inevitably difficult to express such requirements in an unambiguous, accurate and complete manner as should be striven for in order to be able to handle them properly.

- ***Functionality-bias.*** It is a common practice to use abstractions in terms of object types and classes, functions and states from the very beginning. These are certainly well-approved concepts enabling us to cope with (functional) complexity, but are biased towards the functionality dimension. This leads to a functionality-minded development process where extra-functional requirements are degraded to second class citizens which will have no chance to become more than just some attachments to the used abstractions. This might be feasible for some categories of extra-functional requirements like time constraints, but other (multi-faceted) categories of extra-functional requirements such as usability criteria cannot be sufficiently addressed with this strategy.

- ***Contradictions.*** As already shown by the examples of section 2.1., tight interweaved modelling of functional and extra-functional requirements may lead to severe problems in development and maintenance processes. We have to cope with the contradictions which may arise between functional and extra-functional requirements. On the other hand, we have to take into account that the compliance of functional as well as extra-functional requirements will manifest in the final software product. Although a distinction between functional and extra-functional requirements will be feasible especially in early phases of software development, extra-functional requirements, however, will be accomplished in later phases by getting incorporated into 'functional' components of the product to be developed (be it in terms of decisions on the overall architecture or during actual implementations). The moment and manner of this incorporation is the crucial issue in the integration of extra-functional requirements.

The above problems lead us to the following aims which we should try to achieve concerning extra-functional requirements:

- The 'function-biased' focus during the software development process has to be extended to a wider context which also covers extra-functional requirements.
- Too tight interweavings between functional and extra-functional requirements during early phases have to be avoided.
- The integration of extra-functional requirements must take place in a planned and controlled manner.

Potential issues to be addressed in achieving these aims regard capturing and modelling EFRs as well as the integration of EFRs in the software development process. These issues will be subsequently discussed.

EFR Capturing

Processes, methods, techniques and tools currently used in practice did prove to be effective means for supporting the engineering tasks in software development. Nevertheless, real-world experience confronts us with a substantial amount of failed projects, dissatisfied users and unused products (cf. Loucopoulos & Karakostas 1995), all factors which indicate that some essential requirements were not taken into account properly.
Some kinds of extra-functional requirements are fuzzy in their nature and are difficult to handle. This is certainly one reason why these requirements are likely to be neglected or simply forgotten. Additionally, our functionality-biased understanding of software development is all but conducive to turn our attention towards extra-functional requirements. However, adding forgotten functionality to a final software product is far less a challenge than adding quality.
Hence, it is highly advisable to pay adequate attention to extra-functional requirements from the very beginning of the software development process. Although requirements engineering tasks are already considered to be the most crucial activities in software engineering, even the relevance of requirements elicitation alone will increase because of the additional challenge to address extra-functional requirements explicitly.
We all know that a question like "What extra-functional requirements are relevant for this project?" will in most cases be extremely inadequate for getting any useful answer from a stakeholder. It rather requires a systematic EFR-specific preparation of the elicitation process.
It certainly cannot be denied, that extra-functional requirements are highly dependent from the domain which is relevant for the software product. The identification of EFR categories which are of particular interest from the domain point of view will help in collecting elicitation entries for initial questions referring to these categories. These questions will be at a higher level of abstraction though, but are already touching upon potential EFR clusters which might turn out to be worth looking at in detail during elicitation. The usage of the faceted classification schema as introduced in section 2.2. throughout many different development projects will help in finding domain-relevant EFR categories by experience. Once the relationships between domains and EFR categories are identified, templates for each EFR category consisting of category-specific questions will provide a more detailed starting point for EFR-specific elicitation activities.

EFR Modelling

As already mentioned, currently used modelling techniques have realization-focussed abstractions (like types, classes, functions, states) as basic concepts. They deal with structures and behaviors. Complexity can be handled through compositions and different levels of detail. Each modelling technique has its conceptual view, be it structural or behavioral. Different modelling techniques can be used in an additive manner by allowing for multiparadigm connectors in terms of particular concepts (e.g. objects as common concept in static and dynamic modelling).
While functional requirements are dealt with by using abstractions, extra-functional requirements can be (at best) expressed as values attached to process and product attributes (e.g. time constraints). Some categories of extra-functional requirements cannot even be quantified because of their fuzzy and inaccurate nature (e.g. usability); they need additional refinements. The Goal Question Metric (GQM) model (Basili et al. 1994; van Latum et al. 1998) is an example for a method which supports the refinement of (rather abstract) goals by means of questions leading to better quantifyable goals and associated metrics for measurement.
The huge variety and diversity of different categories of extra-functional requirements makes it infeasible to rely on a special single modelling technique which is capable of handling functional and extra-functional requirements together. Trying to extend existing methods used for functional requirements modelling with components for expressing all the different categories of extra-functional requirements would soon overload these methods. As discussed in section 2.1., too tight interweavings of extra-functional with functional

requirements should be avoided in favor of readability, flexibility and maintainability. So far, only few approaches exist which at least enable modelling functionalities together with one isolated dimension of extra-functional requirements, e.g. only performance (Opdahl 1994) or only security (Dobrovnik & Hochmüller 1998).
One approach which particularly focusses on modelling extra-functional requirements is the NFR framework (Chung 1993; Mylopoulos et al. 1992; Chung et al. 1996). It uses goal graphs to represent satisficing decomposition and correlation structures of goals. The system quality model proposed by Harrison et al. (1996) is based on the NFR framework. It refines the notion of correlation in 'supports' and 'promotes' relationships and introduces a template notation for recording possible kinds of quality relationships. One drawback of these methods is their non-applicability in case of complex EFR interrelationships.
The main issue in EFR modelling is to find a balance between the necessity of the representation of relationships between EFRs and to functional requirements and the applicability of methods extended with EFR-specific constructs.

EFR Integration Process

The most crucial challenge is the integration and accomplishment of the proper extra-functional requirements in the appropriate software development phases. Regarding the rumors about the (non-)existence of extra-functional requirements it is no wonder that this problem has not been very extensively tackled by the scientific community so far. The rest of this paper is dedicated to this particular challenge.
For the purpose of EFR integration, two different strategies would be conceivable: a revolutionary or an evolutionary approach.

- The *revolutionary approach* would require to throw overboard our traditional software development strategies. We should have to explicitly avoid the above-mentioned functionality-bias by starting with (functional) abstractions too early in the software life cycle. New methods would be required which at least handle both, functionalities and extra-functionalities, in a like manner. These methods are likely to turn out to be incompatible to each of the currently applied methods. As a consequence, our overall way of thinking would have to be changed radically.

- The *evolutionary approach* would take into account existing development strategies including our preferences regarding abstractions and functionalities. It would accept one driving development process which can be an instance of the currently applied phase-oriented process models. This setting would allow for a smooth integration of extra-functionalities. The main aim of such an approach would be the integration of extra-functional requirements in the right phases. Continuous verification and validation would have to take place in all phases of the development process in order to detect potential inconsistencies (which are most likely to arise - and have grave impact on the whole project - with extra-functionalities rather than in case of mere functionalities) as early as possible.

The advantages of a smooth EFR integration and the possibility to mainly adhere to classical, already approved and repeatedly used development strategies rather call for an evolutionary approach. The concept for such an approach will be outlined in the next section.

EFRs IN THE SOFTWARE DEVELOPMENT PROCESS – AN INTEGRATED VIEW

In the sequel, the role of extra-functional requirements within the software development process will be discussed. Initially, the different EFR categories will be put into relation to the functional dimension of software systems. Then, the different kinds of relationships between these dimensions will be looked at and a general schema for the evolution of extra-functional requirements within the software life cycle will be presented. After outlining the effect of extra-functional requirements on the potential solution space for a software product, the evolutionary process of EFR integration will be introduced by using the cylinder-to-stalactite metaphor.

A Different View of Viewpoints

A software system usually will be a product which is initiated, developed and used by different persons with different concerns. There are not only different perspectives on a definition or characterization of the notion of viewpoints (cf. Leite 1996), but also various approaches tackling the problems of view(point)s in requirements engineering (e.g. Kotonya & Sommerville 1996) and their integration (cf. Nuseibeh et al. 1994). Although some of these approaches already address the integration of viewpoints across different modelling techniques,

common to them is again their functionality focus because of the concentration on structural and/or behavioral view integration.

Regarding our distinction in dimensions of concern, the traditional view of viewpoints which is limited to only one particular category of viewpoints has to be extended to cover also extra-functional dimensions as viewpoint categories on their own. This would lead to a more complete picture which is given in Fig. 3. Classical viewpoint integration mainly takes place within just one viewpoint category. A comprehensive integration approach, however, should also bear in mind this variety of viewpoint categories.

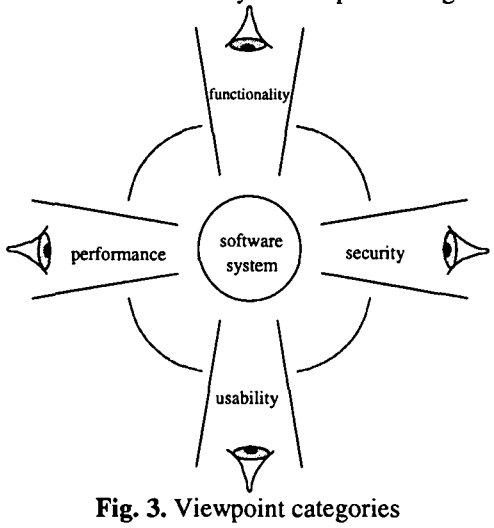

Fig. 3. Viewpoint categories

EFR Relationships and Dynamics

Extra-functional requirements cannot be regarded in isolation. They may constrain the software development process or directly address the software product or its components. Nevertheless, we should be aware of the fact that process constraining requirements (e.g. documentation requirements, given programming language, standards) will finally also manifest themselves in the resulting product.
In general, EFRs are constraining the functional solution space. They may constrain particular functional requirements, compositional parts of the system, the system as a whole or a special contextually coherent part of the system which cannot be directly mapped to a component within the functional scope.
The different categories of extra-functional requirements can neither be regarded in isolation. The fulfillment of some requirements of one EFR category may support or hinder the fulfillment of requirements of another one. Beyond that, situations are possible where single EFRs are not influencing each other, but particular combinations thereof may have different impacts.
As extra-functional requirements have their effects throughout the software development process, conflicts between extra-functional requirements may arise dynamically in all phases of development. This means that some requirements engineering activities (e.g. analysis, negotiations, conflict resolution) which usually are carried out in early phases will have to be effected also during later phases (on demand).
Let us consider a simple example. A flexible and maintainable information system should be developed to be used by various persons of different security levels. Requirements engineering activities include the elicitation of static and dynamic requirements and associated security constraints. The integration of different information requirements result in a conceptual model as structural basis for further design steps. Certain design decisions are required to tackle the security issues. For reasons of better flexibility and maintainability, a strict boundary between the pure semantic parts of the conceptual model and the security-relevant design elements is striven for. This could lead to the decision in favor of a three layered architecture in the sense of ANSI/Sparc using views as security mechanism. This decision may have in turn impacts on other categories of EFRs, e.g. performance problems might arise during the physical design phase. Here we can see, that the accomplishment of a combination of extra-functional requirements of different categories (flexibility, maintainability, security) could cause contradictions to other EFR categories. These contradictions may become evident even only in subsequent development phases. Sometimes, even extra-functional requirements of a new category (e.g. performance requirements) arise and become relevant because of particular results of design decisions in accomplishing existing ones.
Another important aspect is the dichotomy on the border between functional requirements as elicited and analysed during requirements engineering activities and the functional design elements satisficing these requirements. Usually, a one-to-one mapping will not be possible. Therefore, a mere attachment of EFRs to functional requirements is not sufficient.

In general, it is quite reasonable to make a distinction between two different types of extra-functional requirements with respect to their expressiveness:

- ***Crisp EFRs:*** This type of EFRs comprises requirements which can be accurately stated (quantified) as well as checked for accomplishment (measured). The term 'crisp' was chosen to emphasize a certain degree of precision in the formulation of EFRs. As an example for an EFR category which is likely to consist of crisp EFRs, we can assume any kind of time constraints. These constraints will usually concern particular functions and will be stated in terms of time limits.

- ***Vague EFRs:*** Especially when extra-functional requirements start to evolve, they might not be tangible items from the very beginning. Often, they will take shape in a later phase as they may also depend on other decisions made during the development process. Examples for categories of vague EFRs are especially those without directly applicable metrics, like usability, flexibility, maintainability, and so on. Often, the refinement process will require to use *proxies* (e.g. 'easy to learn', 'easy to use' for usability) and associated metrics (e.g. average training period, average amount of help requests) which together serve as a substitute and approximation of the original vague EFR (Wieringa 1996). Vague EFRs are most difficult to handle. Because of their fuzzy nature and our restricted possibilities in grasping them properly, they often tend to be neglected. Additionally, if the refinement process from rather vague to more crisp EFRs was successful, the resulting crisp EFRs might substantially diminish the degrees of freedom for the ensuing software development process.

The evolution of extra-functional requirements within the software development process can be described as follows. EFRs can be stated in terms of vague or crisp EFRs. Vague EFRs have to be refined to result in crisp EFRs. The refinement process to be applied depends on the actual EFR category and the initial EFR statement. Possible means for refinement are additional EFR-specific interviews (e.g. using proxies) or GQM. The decision about the achievement of an acceptable degree of detail is also situation-dependent. The fulfillment of the refined EFRs and the compliance check take place during the rest of the software development process. At any development step before the actual delivery of the software system, it may become obvious that a particular EFR (which already might be fulfilled so far) cannot be met any more. This would require to trigger a conflict resolution process involving certain negotiations. Three kinds of results of this process are possible:

- The EFR is considered to be still important as it is. Adaptation activities will only concern other (conflicting) requirements.
- The EFR is still important but has to be changed because of a compromise. Adaptations to the requirement specification and possibly to already existing software components will be necessary.
- The EFR is abandoned because of other requirements which are more important. Cancellation activities might be necessary.

The state transition diagram of Fig. 4 illustrates these EFR dynamics. This representation slightly deviates from the notation as used in OMT (Rumbaugh et al. 1991). While an arrow from a composite state indicates that the source state can be any of the substates, an arrow to a composite state should indicate that the target state can also be any of the substates (and not a certain initial state), irrespectively of the original source state.

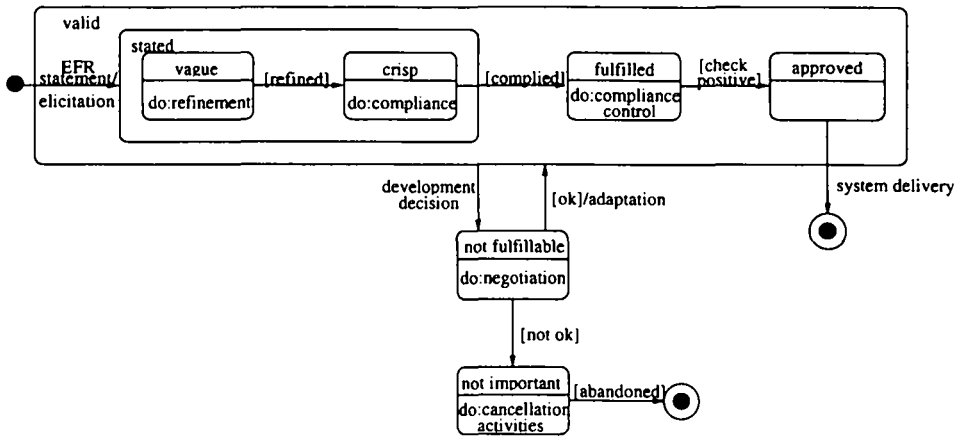

Fig. 4. EFR dynamics and evolution

Fitness for Activation vs. Fitness for Use

Concerning the result of a software development process - the software product - we can assess its quality depending on the spectrum of requirements met by this product.

- ***Fitness for activation.*** A software product which satisfies only functional requirements can be considered as a system fit for activation. It does what it should do in terms of input/output relationships as governed by pure functionality. The maturity of systems fit for activation rather refers to functioning prototypes than to systems of some quality.

- ***Fitness for use.*** A software product which satisfies functional as well as extra-functional requirements is a system fit for use. It does not only do was it should do with respect to input/output relationships but also behaves according to the constraints specified by extra-functional requirements. The notion of fitness for use should not be interpreted in the narrow sense of usability. It rather refers to overall system quality features (cf. Potts 1997(b)).

The solution space for systems fit for activation allows more degrees of freedom in software development. Therefore, it will be larger than the solution space for systems fit for use. Hence, a system fit for use is certainly also a system fit for activation but not necessarily vice versa. A graphical representation of the relationship between both basic solution spaces is given in Fig. 5a.
Leaving it at just the two archetypes of systems would, however, oversimplify the complete picture. As already mentioned, extra-functional requirements may also contradict each other. This would mean that there is no 'ideal' solution space for a system fit for use. Such possible trade-offs between extra-functional requirements have to be managed with adequate negotiations and decision processes. The decisions about preferences and compromise solutions will actually set up an 'agreed' solution space in which a system fit for use should be placed. Fig. 5b illustrates such a situation with three constraint spaces (A, B, C) restricting the original functional solution space. Here, the intersection between A and B was selected to be the agreed solution space.

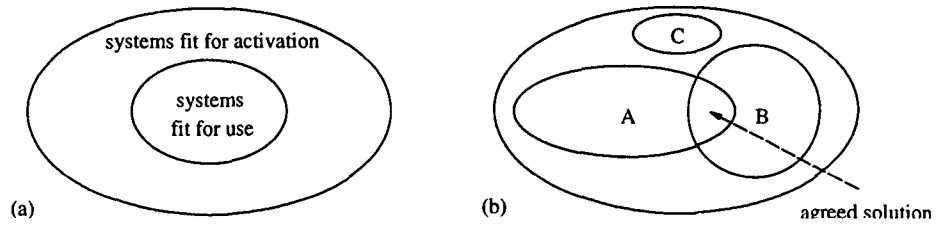

Fig. 5. Solution spaces

The cylinder-to-stalactite metaphor

In the sequel, the metaphor of a cylinder which is tailored to a stalactite (Fig. 6) is used to explain the evolution which takes place during the process of software development. Aiming at an evolutionary approach, we assume a phase-driven process followed for software development which is initially symbolized by an imaginary cylinder. The initial circular base of the cylinder represents the initial solution space which is headed for. The time dimension is partitioned roughly into basic process phases.

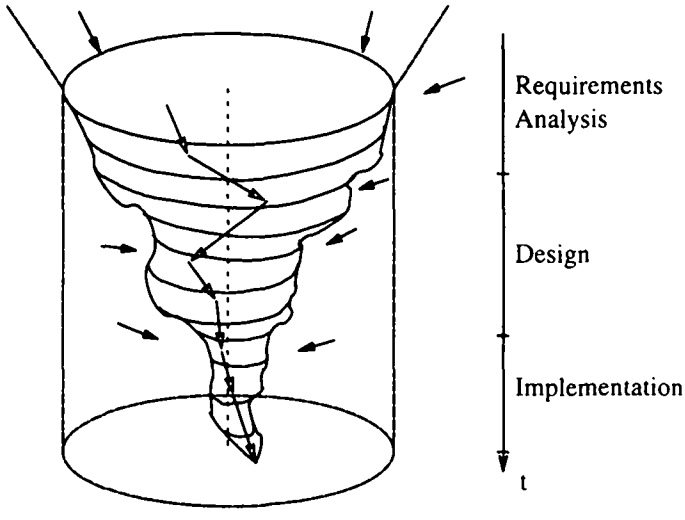

Fig. 6. Cylinder-to-stalactite metaphor

During the phase of requirements analysis, the activities of requirements elicitation, specification, and validation are carried out. The result of this phase is the functional solution space (for systems fit for activation) which embeds an initial solution space constrained by already identified extra-functional requirements.
But the influence of extra-functional requirements does not stop at the end of the requirements analysis phase. Moreover, requirements of different EFR categories have their effects during the whole driving development process (symbolized in Fig. 6 by little arrows from outside). Within the metaphor, we assume that extra-functional requirements restrict the solution space by cutting off pieces from the cylinder. While the shape of the resulting stalactite characterizes the evolution of the solution space within the development period, decisions made during the development process can be observed as directed arrows within the stalactite. At the end of the software development process, the final software product is symbolized by the tip of the stalactite.
The cylinder-to-stalactite metaphor allows us to observe and describe three phenomena:

- The solution space may be partitioned into more than just one part. This can be symbolized by divided stalactites. It requires for decisions choosing the best fitting stalactite to follow.

- Overwhelming EFRs can restrict the solution space in a radical manner such that the tip of the stalactite gets cut off before it has a chance to touch the base of the (initial) cylinder. This would mean that there is no ideal solution possible which meets all the requirements. An immediate conflict resolution is required.

- At any time in the software development process, too big tensions between competing EFRs can lead to negotiations and conflict resolutions. In case of resulting compromises, it might be possible that the already restricted solution space will be expanded again to some extent. Sticking already removed parts back onto the stalactite can symbolize this situation.

At any time within the development process, a sectional view of the stalactite allows for an assessment of the development process. The constrained solution space can be anticipated giving the range in which the final product is likely to be situated. The shape of the stalactite mirrors the sequence of influences originating from extra-functional requirements during the software development process.

EFR INTEGRATION FRAMEWORK

It is rather impossible to think of just one single, fully fledged model which fits perfectly for all software processes. However, many of the frequently used process models are phase-driven. The following ideas on EFR integration aim at the EFR-specific extension of such phase models and are therefore largely independent of the particular activities within the various phases.
A kind of meta framework is proposed to enable the smooth integration of extra-functional requirements into software development processes. This framework concentrates on the fact that extra-functional requirements are eminent throughout a software development process and need continuous attention because of their ongoing influence on the whole process. Principally, we can distinguish between two classes of EFR-related activities within the software development process:

- requirements engineering activities which should also deal with the elicitation, analysis and validation of extra-functional requirements

- further software development activities which are influenced by EFRs and which may in turn affect the degree of EFR compliance

According to this distinction, the EFR integration framework provides EFR-specific hints for early-phase requirements activities but also suggests a reference model for continuously keeping track of extra-functional requirements in later phases of development.

EFR-specific Requirements Engineering

In the types of actual activities to be carried out during requirements engineering, no distinction is necessary between kinds of requirements. Project-relevant requirements - be they functional or extra-functional - have to be elicited, analysed and validated. As already mentioned, our functionality-biased mind as well as our greed for abstractions soon will put our emphasis on functional requirements leading to behavioral and structural requirements models.
Being thoroughly convinced, however, that extra-functional requirements are at least as important as functional ones and that these requirements have to be treated properly too, is already a fertile ground for openness regarding additional effort for EFR-specific analysis activities. The early identification of the set of extra-functional requirements constraining the potential solution space for the software product is certainly the most crucial challenge.
The following suggestions for identifying extra-functional requirements will not address particular EFR categories nor give any advice in EFR specification. The purpose is just to give some general practical hints for their explicit consideration.
A first information about extra-functional requirement categories which might be of importance for the software product can be obtained from analysing the problem domain. Some of these domain-relevant EFR categories may be conflicting giving an initial conflict set at category level. Elicitation of project-specific priorities regarding these categories will result in additional information about the potential conflict proportions.
The actual instances of project-relevant EFR categories have to be identified during the elicitation process in terms of vague and crisp extra-functional requirements (cf. section 4.2.). Vague requirements need further refinement (as described in section 4.2.) as soon as possible as it is easier to identify potential conflicts between crisp requirements than in cases when vague requirements are involved.
Upon identification of conflicting extra-functional requirements, negotiations become inevitable. All decisions resulting from these negotiations must be made evident and documented to be able to be referenced later on.
The additional role of an EFR advocate (Hochmüller 1997(b)) can help in explicitly focussing on extra-functional requirements during requirements engineering and the rest of the software development process.

Question - Plan – Action

For a smooth integration of extra-functional requirements into subsequent development phases, we propose the *Question Plan Action* (*QPA*) approach. QPA does not explicitly address special categories of extra-functional requirements neither, but rather enables us to identify milestones where extra-functional requirements particularly need our consideration.
Each phase-oriented software development process can be extended by QPA. For such an extension, the underlying process model is assumed to dispose of the following granules:

- process phase
- development step within a phase
- decision making process

A process phase is a period of time in which a particular kind of development task is fulfilled. It has a certain entry point at which the phase begins and a certain exit point which means that all the activities foreseen for the phase were carried out. The elementary activities within a phase correspond to so-called development steps (Fig. 7). Decisions are particular development steps which influence subsequent activities of the same or later phases.

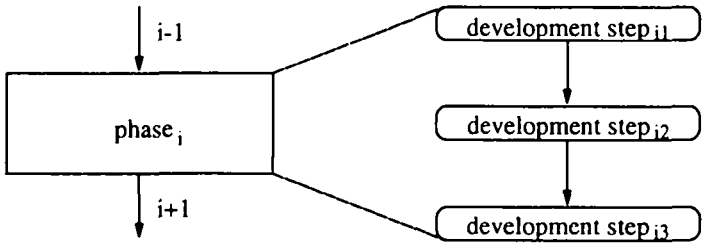

Fig. 7. Process granules

The structure of the QPA approach can be outlined as follows:

- ***Question (Q):*** Special questions referring to EFRs serve as connection points between the driving software development process and extra-functional requirements. Their purpose is the identification of particular extra-functional requirements which are of importance for the actual process granule.

- ***Plan (P):*** Depending on the answers to the questions, the appropriate actions regarding the involved extra-functional requirements have to be planned and scheduled. A plan can also define a new project-dependent question which can be scheduled to be asked at a later stage of development either within the same granule or within a subsequent one.

- ***Action (A):*** The actions actually carry out the planned EFR integration. These actions will take place in terms of development steps.

In general, an answer to a question will be immediately followed by the definition of a plan. The action carrying out the plan usually will be executed at a later time.

There are two kinds of questions:

- ***Predefined meta questions:*** These questions are general enough to be applicable in any project regardless of the actual development project. The rest of this section is dedicated to give an overview of these questions which are stated in analogy with the evolution of EFRs (cf. sec. 4.2.).

- ***Project-specific questions:*** In addition to the rather generally stated meta questions, the QPA approach also allows for questions which can be defined on demand. These questions will be formulated specific to the actual project.

QPA for Process Phases

Regarding a phase as elementary unit in software development, there are two possible points of connection for QPA: the entry and the exit of a phase.

The meta questions to be answered at the beginning and the end of a phase refer to already identified extra-functional requirements and their different states (cf. sec. 4.2.).

Especially in case of process phases, actions are of pure operative nature. Hence, the following QPA description will be confined to the identification of connection points together with the relevant meta questions and the plans to be developed.

- ***on entry***

The following meta questions are proposed at the very beginning of each phase. For each question, a rough sketch about the contents of the associated plan is outlined.

Q_{entl}: What EFRs have to be directly obeyed in this phase?
P_{entl}: Plan for continuous observation of phase relevant EFRs
For each identified EFR:
- When has it to be obeyed?
- Who has to obey it?
- How should it be obeyed?

Q_{ent2}: What EFRs have to be fulfilled in this phase?
P_{ent2}: Plan for EFR fulfillment
For each identified EFR:
- In which development step(s) should it be fulfilled?
- When should the fulfillment be checked (earliest, latest date)?
-

Q_{ent3}: What EFRs have to be checked in this phase for compliance?
P_{ent3}: Plan for EFR compliance check
For each identified EFR:
- In which development step should the compliance be checked?
- Who is responsible for the check?

- ***on exit***
  After having carried out the activities of one development phase but before passing over to the next phase, the following questions are relevant:

Q_{ex1}: What EFRs of the set identified in Q_{ent1} have not been obeyed in this phase?
P_{ex1}: Plan for assessment of effects of neglecting these EFRs
For each identified EFR:
- Why have they not been obeyed?
- Which effects (risks) could/did arise?
- Decision about further steps.

Q_{ex2}: What EFRs of the set identified in Q_{ent2} have not been fulfilled in this phase?
P_{ex2}: Plan for assessment of effects of not fulfilling these EFRs
For each identified EFR:
- Which effects (risks) could/did arise?
- Decision about further steps; eventually immediate fulfillment.

Q_{ex3}: What EFRs of the set identified in Q_{ent3} did fail during check?
P_{ex3}: Plan for assessment of failed check
For each identified EFR:
- Which effects (risks) could/did arise?
- Decision about further steps; eventually immediate fulfillment and repeated check.

QPA for Development Steps

Within each development step, actual EFR-related actions have to be planned and carried out. The difference to the QPA elements for process phases lies in the granularity of the plans leading to actual actions.

Q_{dev1}: Which EFRs have to be addressed in the step?
P_{dev1}: Plan actual treatment (fulfillment, continuous observation)

Q_{dev2}: Which EFRs are influenced by the actions in the development step?
P_{dev2}: Plan the assessment of how these EFRs are influenced

QPA for Decisions

A decision making process involves decisions which might be influenced by EFRs or which in turn may influence certain EFRs. As decision making processes usually do not last a very long time, actions are likely to follow immediately upon answering the questions.

Q_{dec1}: Which EFRs influence the decision?
A_{dec1}: Continuously observe these EFRs during decision making process

Q_{dec2}: Which EFRs are influenced by the decision in what aspect?
A_{dec2}: Analyse and evaluate these influences.

Q_{dec3}: Which conflicts become evident during decision making process?
A_{dec3}: In case of conflicts:

- Negotiate
- Resolve conflicts (within actual decision making process)
- Document the decision making process (and the changed/new EFRs)
- Document the results

Representation Considerations

As the Question Plan Action approach inherently influences and requires the coordination of EFR-relevant activities throughout the whole software development process, appropriate support in the administration of those activities together with the necessary QPA steps is desired. Such a support should provide predefined meta questions and help in stating project-specific questions on demand as well as in expressing plans. Generally, it should be possible to establish links between corresponding QPA elements as well as between software development granules and QPA elements.
In addition to the assistance in integrating QPA steps into the software development process, also the administration and representation of extra-functional requirements, their relationships and interdependencies need appropriate support. Usually, the multi-dimensional structure of all these relationships will be rather complex and difficult to survey. Hence, any attempt to get along with just one universal graphical representation keeping track of all the details is likely to fail. Similarly, it is useless to apply just a linear textual description.
Hypertext concepts with different types of links between different types of nodes (including graphics) will certainly better meet the representation requirements outlined above. The benefits of semiformal hypertext as means to support requirements engineering were already demonstrated (e.g. RETH (Kaindl 1993), HYDRA (Pohl & Haumer 1995)). For our purposes, hypertext will not only be utilized to structure requirements information during requirements engineering. It likewise will serve as a mechanism to relate this information to software development activities throughout the development process. This enables the navigation between process elements and respective EFR information.
While the hypertext paradigm is suitable as a presentation mechanism, the amount of data together with multiple concurrent access and update operations call for a consistent and powerful database for EFR information management. For example, the EFR information base (Hochmüller 1997(b)) helps in keeping track of extra-functional requirements, their origins, relevance, (inter)relationships and compliances. It can provide the data required for QPA and can also be used for EFR-related documentation purposes.

Usefulness of the EFR Integration Framework

The main goal of the EFR integration framework is its applicability in a wide variety of traditional phase-driven development processes. We explicitly waived the introduction of any new formalisms in favor of the emphasis on crucial facts and activities dealing with EFRs. So, software development units (teams, enterprises) can decide on their own how much effort they are willing to pay for the treatment of actual EFRs. The effort will depend on the perceived risks emerging from particular EFRs.
Its intensity can range from developing and using simple checklists to the development of own methods for dealing with particular EFR categories (e.g. simulation in case of performance requirements) together with the reuse of project experiences for continuous refinement and adaptation of the integration framework (e.g. additional domain-specific questions for future project within the same problem domain). Tool support should be taken into consideration even in case of rather simple methods (like checklists) since they will enhance productivity of and overall method acceptance by project members as well as improve the development process by better traceability and documentation of EFR-related activities.

RELATED WORK – COMPARISON AND SYNERGIES

The presence of the computer science community's disagreements regarding the existence and taxonomy of extra-functional requirements is already an indicator for the lack of substantial relevant research results. However, there are few approaches which also tackle the issue of integrating extra-functional requirements into the software development process.
The approach by Dobson & McDermid (1991) proposes two process-related reference models which describe the different facets of the overall software development process. One of them assumes a canonical development stage and outlines the activities of analysis, evaluation and synthesis and their input-output relationships to policies and external constraints. The interrelationship to any conventional phase-based model, however, is not explicitly addressed at all. The other reference model of Dobson & McDermid (1991) considers the different facets of a specification (in terms of the 4 roles of (extra-functional) requirements) within the development process and how they relate at different life cycle stages. Although a distinction between extra-functional and functional requirements is avoided, its main problem - the potentially complex interplay between both types of requirements leading to that rather unsatisfactory dichotomy - remains which becomes evident in the attempt to

find the appropriate categorization for classic EFRs in terms of commitments or obligations. The main drawback of this approach is the missing managerial aspect of EFR integration.
The goal-oriented NFR framework (Chung 1993; Mylopoulos et al. 1992; Chung et al. 1996; Nixon 1993) has its origin in the context of information systems design. It deals with the decomposition of goals but does not address the integration of extra-functional requirements into the overall software development process. However, it serves as basis for another process-oriented approach dealing with extra-functional requirements integration. MIKE (Model-based Incremental Knowledge Engineering) is a framework for the development of knowledge-based systems which explicitly treats extra-functional requirements within its proposed process model (Laudes & Studer 1995). Extra-functional requirements are described semiformally as NFR context in a hypermedia-based fashion. The impact of extra-functional requirements on decisions taken in the design phase is explicitly described in a particular process model. While the NFR framework mainly deals with the decomposition of goals and stops at a higher level of granularity, MIKE also handles elementary design decisions.
The QPA framework tries to glue extra-functional requirements on a given phase-driven software development process. It is independent of any project domain and the actual activities carried out within the process. Nevertheless, the phase dealing with requirements engineering will include special activities for EFR capture and modelling. The related tasks can be fulfilled by applying existing approaches. Methods like GQM (Basili et al. 1994; van Latum et al. 1998) or other quality measures (Gillies 1992) can be used to identify and quantify high-level EFRs (goals) and to proceed from vague to crisp EFRs.
Synergies can also be achieved by applying organization-specific requirements management with particular focus on extra-functional requirements. An EFR information base (cf. section 5.3.) which is used throughout the software development process will additionally support traceability of EFRs back to their origins and rationales and forward to their compliances.

CONCLUSION

Based on two rather simple examples in modelling synchronization constraints in OOCP and security constraints in OODBMSs, the existence of an 'extra' dimension of requirements was demonstrated and the problems connected with tight interweaved treatment of functional and extra-functional requirements were pointed out.
Beside the issues of capturing and modelling extra-functional requirements, the integration of this kind of requirements into the software development process was identified as the most crucial challenge. Systems which should be fit for use call for such an integration.
This paper presented an evolutionary approach for the integration of extra-functional requirements in an overall phase-driven development process. This EFR integration framework is universal and flexible in the sense, that it can supplement a wide range of phase models currently in use. Additional benefits can be achieved by synergies from combining this framework with other approaches (like the management of extra-functional requirements) or techniques (e.g. hypertext representation of multidimensional EFR structures and process documentation for better traceability of negotiations and decisions).

REFERENCES

Basili VR, Caldiera G., Rombach HD. Goal Question Metric Approach. In: Marciniak JJ (ed). **Encyclopedia of Software**. vol 1. 1994. pp 528-532

Bertino E, JajodiaS. Modeling multilevel entities using single level objects. In: Ceri S, Tanaka K, Tsur S (eds). **Deductive and object-oriented databases (DOOD'93)**. Springer. LNCS 760. pp 415-428

Bertolino A. Achieving quality in software. **Journal on Systems and Software** 1994; 26(1): p 13

Boehm BW. Software engineering. **IEEE Transactions on Computers** 1976; 25(12): pp 1226-1241

Boehm BW, In H. Identifying quality requirements conflicts. **IEEE Software** 1996; 13(2): pp 25-35

Chandrasekaran B, Goel AK, Iwasaki Y. Functional representation as design rationale. **IEEE Computer** 1993; 26(1): pp 48-56

Chen P. The entity-relationship model: toward a unified view of data. **ACM Transactions on Database Systems**, 1976; 1(1): pp 9-36

Chung L. Dealing with security requirements during the development of information systems. **Proceedings of the fifth international conference, CAiSE'93**, Paris, 8-11 June 1993. Springer-Verlag, Berlin, 1993. pp 234-251

Chung L, Nixon BA, Yu E. Dealing with change: an approach using non-functional requirements. **Requirements Engineering Journal.** 1996; 1(4): pp 238-260

Coad P, Yourdon E. **OOA - object-oriented analysis.** Prentice-Hall, Englewoods Cliffs, 1991

Davis AM. **Software requirements: Objects, Functions, and States.** 2nd edn. Prentice-Hall, Englewoods Cliffs, 1993

Dobrovnik M, Hochmüller E. Views as security mechanism in object-oriented database systems. **Proceedings of the international workshop on issues and applications of database technology (IADT'98)**, July 1998

Dobson JE, McDermid JA. An investigation into modelling and categorisation of non-functional requirements (for the specification of surface naval command systems). **TR 320**, Computing Laboratory, University of Newcastle upon Tyne, November 1991

Dubois E, Opdahl AL, Pohl K. REFSQ'97 Workshop Summary, **Proceedings of the third international workshop on requirements engineering: foundations of software quality (REFSQ'97)**, Barcelona, Spain, 16-17 June 1997. Presses universitaires de namur, pp 1-12

Gillies AC. Software quality - theory and management. Chapman & Hall, London, 1992

Harrison R, Pratten CH, Sivess V. Quality decomposition for large-scale systems". Technical Report, Department of Electronics and Computer Science, University of Southampton, May 1996

Hochmüller E. Requirements classification as a first step to grasp quality requirements. **Proceedings of the third international workshop on requirements engineering: foundations of software quality (REFSQ'97)**, Barcelona, Spain, 16-17 June 1997(a). Presses universitaires de namur, pp 133-144

Hochmüller E. Quality improvement through quality requirements management. **Proceedings of the international conference on software quality (ICSQ'97)**, Maribor, November 1997(b), pp 71-78

Hochmüller E. Inheritance contradictions between functional and extra-functional requirements. **Journal of integrated design and process science**. SDPS. June 1998; 2(2): pp 22-31

IEEE Recommended Practice for Software Requirements Specifications. **ANSI/IEEE Std. 830-1993. IEEE**, New York, 1993

ISO/IEC. **International standard ISO/IEC 9126**. Information technology software product evaluation - quality characteristics and guidelines for their use. Geneva, 1991

Jackson M. **System development.** Prentice-Hall, 1983

Jajodia S, Kogan B, Sandhu RS. A multilevel secure object-oriented data model. In: Abrams MD, Jajodia S, Podell H.J (eds). **Information security - an integrated collection of essays**. IEEE-CSP. 1995. pp 596-616

Kaindl H. The missing link in requirements engineering. **ACM Software Engineering Notes** 1993; 18(2): pp 30-39

Kaindl H. An integration of scenarios with their purposes in task modeling. **Proceedings of the ACM Symposium on Designing Interactive Systems (DIS'95)**, Ann Arbor, MI, August 1995, pp 227-235

Keller SE, Kahn LG, Panara RB. Specifying software quality requirements with metrics. In: Thayer RH, Dorfman M (eds). **System and software requirements engineering**. IEEE-CSP Tutorial. 1990 pp

Kotonya G, Sommerville I. Requirements engineering with viewpoints. **Software Engineering Journal** 1996; 11(1): pp 5-18

Kramer J, Wolf AL. Succeedings of the 8th international workshop on software specification and design. **ACM Software Engineering Notes** 1996; 21(5): pp 21-35

Laudes D, Studer R. The treatment of non-functional requirements in MIKE. **Proceedings of the 5th european software engineering conference (ESEC'95)**, Sitges, September 1995, pp 294-306

van Latum F, van Solingen R, Oivo M, Hoisl B, Rombach D, Ruhe G. Adopting GQM-based measurement in an industrial environment. **IEEE Software** 1998; 15(1): pp 78-86

Leite JC. Viewpoints on viewpoints. **Proceedings of the international workshop on multiple perspectives in software development (Viewpoints 96)**, San Francisco, October 1996

Loucopoulos P, Karakostas V. **System requirements engineering**. McGrawHill, New York, 1995

Matsuoka S, Yonezawa A. Analysis of inheritance anomaly in object-oriented concurrent programming languages. In: Agha G, Wegner P, Yonezawa A (eds). **Research directions in concurrent object-oriented programming.** MIT Press, 1993, pp. 107-150

Mittermeir RT. Requirements Engineering. In: Kurbel K, Strunz H. Handbuch **Wirtschaftsinformatik**. Poeschel Verlag, 1990

Mittermeir RT. Dimensions of software design - from algorithms to systems. **Proceedings of the second world conference on integrated design & process technology (IDPT'96)**, vol 1, SDPS, Austin, December 1996, pp 82-89

Mylopoulos J, Chung L, Nixon B. Representing and using non-functional reqirements: a process-oriented approach. **IEEE Transactions on Software Engineering** 1992; 18(6): pp 483-497

Nixon BA. Dealing with performance requirements during the development of information systems. Proceedings of the **IEEE international symposium on requirements engineering (RE'93)**, San Diego, 4-6 January 1993, pp 42-49

Nuseibeh B, Kramer J, Finkelstein A. A framework for expressing the relationships between multiple views in requirements specification. **IEEE Transactions on Software Engineering** 1994; 20(10): pp 760-773

Opdahl A. Requirements engineering for software performance. **Proceedings of the first international workshop on requirements engineering: foundations of software quality (REFSQ'94)**, Utrecht, Netherland, 6-7 June 1994, pp 16-32

Pohl K, Haumer P. HYDRA: A hypertext model for structuring informal requirements representations. **Proceedings of the second international workshop on requirements engineering: foundations of software quality (REFSQ'95),** Jyväskylä, Finland, 12-13 June 1995, pp 118-134

Potts C. Requirements models in context. **Proceedings of the third IEEE international symposium on requirements engineering (RE'97),** Annapolis, Maryland, 6-10 January 1997(a), pp 102-104

Potts C. Fitness for use: the system quality that matters most. **Proceedings of the third international workshop on requirements engineering: foundations of software quality (REFSQ'97),** Barcelona, Spain, 16-17 June 1997(b). Presses universitaires de namur, pp 15-27

Prieto-Diaz R. Implementing faceted classification for software reuse. **Communications of the ACM** 1991; 34(5): pp 88-97

Rai A, Song H, Troutt M. Software quality assurance: an analytical survey and research prioritization **Journal on Systems and Software** 1998; 40(1): pp 67-83

Roman GC. A taxonomy of current issues in requirements engineering, **IEEE Computer** 1985; 18(4): pp 14-23

Ross DT, Schoman KE. Structured analysis for requirements definition. **IEEE Transactions on Software Engineering** 1977; 3(1): pp 1-65

Rumbaugh J, Blaha M, Premerlani W, Eddy F, Lorensen W. **Object-oriented modelling and design.** Prentice-Hall, Englewoods Cliffs, 1991

Rumbaugh J, Jacobson I, Booch G. **The Unified Modeling Language Reference Manual.**Addison-Wesley, 1998

Shaw M. Truth vs knowledge: the difference between what a component does and what we know it does. **Proceedings of the eighth international workshop on software specification and design (IWSSD-8),** Schloss Velen, Germany, March 1996, pp 181-185

Sommerville I. **Software engineering.** Addison-Wesley, 1992

Starke G. Session summary: non-functional requirements. **Proceedings of the first international workshop on requirements engineering: foundations of software quality (REFSQ'94),** Utrecht, Netherland, 6-7 June 1994, pp 4-6

Wieringa RJ. **Requirements engineering - frameworks for understanding.** John Wiley & Sons, Chichester, 1996

Yourdon E, Constantine LL. **Structured design.** Prentice-Hall, 1979

Yourdon E. **Modern structured analysis.** Prentice-Hall, 1989