On Non-Functional Requirements in Software Engineering

Lawrence Chung[1] and Julio Cesar Sampaio do Prado Leite[2]

[1] Department of Computer Science, The University of Texas at Dallas
[2] Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro
www.utdallas.edu/~chung/, www.inf.puc-rio.br/~julio

Abstract. Essentially a software system's utility is determined by both its functionality and its non-functional characteristics, such as usability, flexibility, performance, interoperability and security. Nonetheless, there has been a lop-sided emphasis in the functionality of the software, even though the functionality is not useful or usable without the necessary non-functional characteristics. In this chapter, we review the state of the art on the treatment of non-functional requirements (hereafter, NFRs), while providing some prospects for future directions.

Keywords: Non-functional requirements, NFRs, softgoals, satisficing, requirements engineering, goal-oriented requirements engineering, alternatives, selection criteria.

1 Introduction

"Soft is harder to deal with than hard." [Anonymous]

Essentially a system's utility is determined by both its functionality and its non-functional characteristics, such as usability, flexibility, performance, interoperability and security. Nonetheless, there has been a lop-sided emphasis in the functionality of the system, even though the functionality is not useful or usable without the necessary non-functional characteristics.

Just with almost everything else, the concept of quality is also fundamental to software engineering, and both functional and non-functional characteristics must be taken into consideration in the development of a quality software system. However, partly due to the short history behind software engineering, partly due to the demand on quickly having running systems fulfilling the basic necessity, and also partly due to the "soft" nature of non-functional things, most of the attention in software engineering in the past has been centered on notations and techniques for defining and providing the functions a software system has to perform.

A frequently observable practice, as a result of this lop-sided emphasis in the functional side of a software artifact, is that the needed quality characteristics are treated only as technical issues related mostly to the detailed design or testing of an

A.T. Borgida et al. (Eds.): Mylopoulos Festschrift, LNCS 5600, pp. 363–379, 2009.

implemented system. This kind of practice, of course, is quite inadequate. Detailed design and testing do not make much sense without their preceding phases of understanding what the real-world problem is to which a software system might be proposed as a solution and also what the specifics of the software solution, i.e., the requirements, might be like. And real-world problems are more non-functionally oriented than they are functionally oriented, e.g., poor productivity, slow processing, high cost, low quality, and unhappy customer.

Although the requirements engineering community has classified requirements as either functional or non-functional, most existing requirements models and requirements specification languages lacked a proper treatment of quality characteristics. Treating quality characteristics as a whole, and not just as functionality alone, has been a key focus of works in the area of goal-oriented requirements engineering [1] [2], and in particular the NFR Framework [3] that treats non-functionality at a high level of abstraction for both the problem and the solution.

This chapter brings forth a review of the literature on NFRs, with emphasis in the different definitions, representation schemes, as well as more advanced uses of the concepts. At the end, we conclude the chapter by discussing open issues in the early treatment of NFRs and its impacts on software construction.

2 What are Non-Functional Requirements?

In literature, a plethora of definitions can be found of non-functional requirements (NFRs).

Colloquially speaking, NFRs have been referred to as "-ilities" (e.g., usability) or "-ities" (e.g., integrity), i.e., words ending with the string "-ility" or "-ity. A large list of such words can be found, for example, in [3]. There are many other types of NFRs that do not end with either "-ility" or "-ity" as well, such as performance, user-friendliness and coherence.

An important piece of work on NFRs is the NFR Framework [1] [3], which decouples the concept of functionality from other quality attributes and concerns for productivity, time and cost, by means of a higher-level of abstraction. Instead of focusing on expressing requirements in terms of detailed functions, constraints and attributes, the NFR Framework devised the distinction of NFRs by using the concepts of goal and softgoal. More details on the NFR Framework will be described further in Section 4.

In the area of Software Architecture, one frequently encountered keyword is "quality attributes" [4], which is understood as a set of concerns related to the concept of quality. For a definition of quality, an IEEE standard [5] is used here as a companion: "Software quality is the degree to which software possesses a desired combination of attributes (e.g., reliability, interoperability)."

Several other authors have also treated these types of concerns. For instance, basic quality (functionality, reliability, ease of use, economy and safety) is distinguished from extra quality (flexibility, reparability, adaptability, understandability, documentation and enhanceability) in [6].

In the area of engineering and management, the well known QFD (Quality Function Deployment) strategy [7] distinguishes positive quality from negative quality: "QFD is quite different in that it seeks out both "spoken" and "unspoken" customer requirements and maximizes "positive" quality (such as ease of use, fun, luxury) that creates value. Traditional quality systems aim at minimizing negative quality (such as defects, poor service)". One of the techniques used by QFD strategies is the House of Quality [8], in which the process starts "...with the customer, whose requirements are called customer attributes (CA´s) - phrases customers use to describe products and product characteristics...". Incidentally, none of the examples of the CA´s in [8] is related to functionality or just functionality alone.

In the area of software requirements, the term non-functional requirements [9] has been used to refer to concerns not related to the functionality of the software. However, different authors characterize this difference in informal and unequal definitions. For example, a series of such definitions is summarized in [10]:

> a) "Describe the non-behavioral aspects of a system, capturing the properties and constraints under which a system must operate. "
> b) "The required overall attributes of the system, including portability, reliability, efficiency, human engineering, testability, understandability, and modifiability."
> c) "Requirements which are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet."
> d) "... global requirements on its development or operational cost, performance, reliability, maintainability, portability, robustness, and the like. ... There is not a formal definition or a complete list of nonfunctional requirements."
> e) "The behavioral properties that the specified functions must have, such as performance, usability."
> f) "A property, or quality, that the product must have, such as an appearance, or a speed or accuracy property."
> g) "A description of a property or characteristic that a software system must exhibit or a constraint that it must respect, other than an observable system behavior."

After arguing that the definitions are unclear and that they lack consensus, the author says: "For persons who do not want to dispose of the term 'non-functional requirement', we can define this term additionally as: DEFINITION. A non-functional requirement is an attribute of or a constraint on a system." [10].

There are other additional definitions in literature worth adding to the list.

> h) " … types of concerns: functional concerns associated with the services to be provided, and nonfunctional concerns associated with quality of service – such as safety, security, accuracy, performance, and so forth." [2].
> i) "The term "non-functional requirement" is used to delineate requirements focusing on "how good" software does something as opposed to the functional requirements, which focus on "what" the software does." [11].

> j) "Putting it another way, NFRs constitute the justifications of design decisions and constrain the way in which the required functionality may be realized." [12].

On purpose, we left the citation to [12] as the last definition of the several presented. This definition of nonfunctional requirements is of major importance and will be commented later on in Section 4.

Since we are revisiting so many definitions, it might help to focus on the definition of four words that seem central to all of the definitions: quality, functionality, functional and nonfunctional. The definitions were selected from WordNet [13]. WordNet is a lexical database of English, where nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Here we list the major definition of each of these words as they appear in Wordnet. We do this to call the attention to the term nonfunctional.

Quality: Noun -- S: (n) quality (an essential and distinguishing attribute of something or someone)

Functional: Adjective -- S: (adj) functional (designed for or capable of a particular function or use)

Nonfunctional: Adjective -- S: (adj) nonfunctional (not having or performing a function); S: (adj) malfunctioning, nonfunctional (not performing or able to perform its regular function)

Functionality: Noun -- S: (n) functionality (capable of serving a purpose well)

If we carefully examine these definitions, we notice that the word "functional" is an adjective and "quality" is both noun and adjective, whereas "functionality" is a noun. We also notice that "functional" refers to use and "functionality" refers to purpose.

We bring these definitions to bear, since if we understand the terms "functional requirements" and "non-functional requirements" out of context, they may bring up different semantics. Of course, the term "non-functional requirements" is not meant to mean requirements that are not able to perform a function, but if interpreted out of context, it may create a confusion. If the literature were more careful in choosing names, this potential confusion could have been avoided.

Notwithstanding these observations and the fact that functionality can be seen as quality as well, as also note in [6], we understand that the distinction among these two types of quality is extremely helpful and important in software engineering.

A central point for the distinction of functionality and other qualities is that, for software construction, the purpose of the software system needs to be well defined in terms of the functions that the software will perform. It may sound strange, but this distinction is not as evident in other areas of engineering, as we could see from the QFD strategy. In business and engineering, the function of an artifact or an activity mostly deals with physical entities, and is usually clear and upfront. In contrast, in software engineering whose products are conceptual entities, this is not the case. As a matter of fact, it would be odd to detail, by functions, the fact that a car has to be able to transport people, but, for a software system to be built, a software engineer has to understand what functions it should perform, usually with greater difficulty since they are not evidently visible, measurable, touchable, etc. Furthermore, software is so

rapidly being applied to new application areas that it is not possible for a software engineer to build always on experiences. It is a rather well known fact that a software product may be targeting a domain not familiar to a software engineer – a problem that other types of engineers usually do not have to confront with.

The distinction between functionality and other qualities in the field of requirements engineering has an important benefit: it makes clear to software engineers that requirements are meant to deal with quality attributes and not with just one of them. As the software industry became more mature and different domains were explored by software engineers, it became clearer that it would not be enough just to deal with the description of the desired functionality, but that quality attributes should be carefully thought of early on as well.

So, in the presence of so many different definitions on NFRs, how should we proceed? We want our working definition to be as consistent with, and accommodating, other definitions. As a working definition, we start with the colloquial definition of NFRs, as in the NFR framework [1] [3], namely, any "-ilities", "-ities", along with many other things that do not necessarily end with either of them, such as performance, user-friendliness and coherence, as well as concerns on productivity, time, cost and personal happiness. In view of mathematical functions, in the form of,

$$f: I \rightarrow O \text{ (e.g., sum: int x int } \rightarrow \text{int)},$$

just about anything that addresses characteristics of *f, I, O* or relationships between *I* and *O* will be considered NFRs. For example, whether the summation function can easily be found on a calculator, whether the function can easily be built or modified, in a time- and cost-effective manner, whether the function returns the output fast, who can see the function, the inputs, or the output, for instance.

3 Some Classification Schemes

As seen in the previous section, various pieces of work provide for ways to distinguish among different types of quality concerns. One is the distinction between basic and extra quality [6]. Another is the distinction among concerns (sub-attributes of a quality attribute), factors (or impairments - possible properties of the system, such as policies and mechanisms built into the system, that have an impact on the concerns) and methods (means used by the system to attain the concerns) [4].

The standard ISO/IEC 9126 [14] is also noteworthy which distinguishes 4 types of quality levels: quality in use, external quality, internal quality and process quality. Based on these types, [11] provides a process oriented classification comprised of :

1) "The identification of NFR from different viewpoints and different levels of detail."
2) "The support for uncovering dependencies and conflicts between them, and to discuss and prioritize them accordingly."
3) "The documentation of NFR and the evaluation of this documentation."
4) "The support for identifying means to satisfy the NFR, to evaluate and discuss means, and to make trade-off decision accordingly. This includes cost estimation.", and
5) "The support for change and project management."

Another proposal is made in [15], using the concepts of the NFR Framework [3], on a classification of goals and softgoals, driven by the "non functional perspective". This classification provides 4 categories: functional hardgoals, nonfunctional hardgoals, functional softgoals and nonfunctional softgoals.

Another classification scheme is introduced in [16]:

- Interface requirements: describe how the system is to interface with its environment, users and other systems. E.g., user interfaces and their qualities (e.g., user-friendliness).
- Performance requirements: describe performance constraints involving
  - time/space bounds, such as workloads, response time, throughput and available storage space. E.g., "system must handle 100 transactions/second."
  - reliability involving the availability of components and integrity of information maintained and supplied to the system. E.g., "system must have less than 1hr downtime/3 months."
  - security, such as permissible information flows.
  - survivability, such as system endurance under fire, natural catastrophes.
- Operating requirements: include physical constraints (size, weight), personnel availability, skill level considerations, system accessibility for maintenance, etc.
- Lifecycle requirements: can be classified under two subcategories:
  - quality of the design: measured in terms such as maintainability, enhanceability, portability.
  - limits on development, such as development time limitations, resource availability, methodological standards, etc.
- Economic requirements: immediate and/or long-term costs
- Political requirements

Figure 1 depicts a software quality tree [17] which aims to address concerns for key types of NFRs and importantly possible correlations among them.

FURPS is an acronym representing a model for classifying software quality attributes or non-functional requirements, developed at Hewlett-Packard, and + was later added, hence FURPS+, to extend the acronym to emphasize various attributes [18]:

- Functionality - Feature set, Capabilities, Generality, Security
- Usability - Human factors, Aesthetics, Consistency, Documentation
- Reliability - Frequency/severity of failure, Recoverability, Predictability, Accuracy, Mean time to failure
- Performance - Speed, Efficiency, Resource consumption, Throughput, Response time
- Supportability - Testability, Extensibility, Adaptability, Maintainability, Compatibility, Configurability, Serviceability, Installability, Localizability, Portability

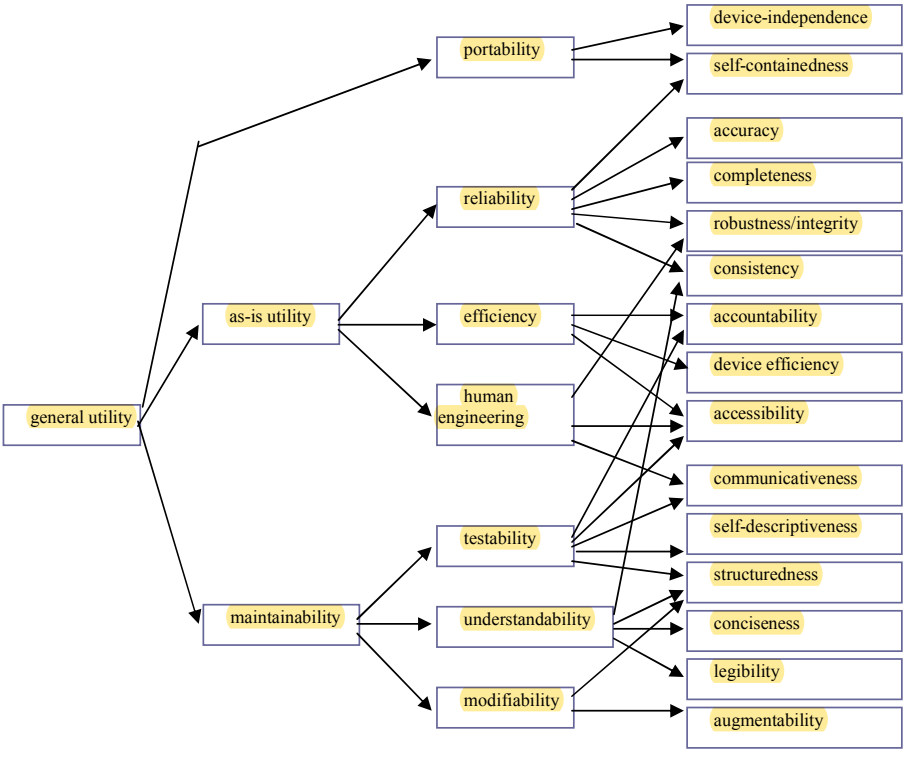

Fig. 1. Software Quality Tree [17]

However, even these well-known classification schemes are inconsistent with each other. For example, consider performance. In Roman's classification scheme, it is defined in terms 4 sub-categories, whereas it does not even appear in software quality tree, while it is shown but quite differently in FURPS+.

Another observation is that neither Roman's nor FURPS+ recognizes any potential interactions among NFRS, while software quality tree does so to a certain extent. For example, in software quality tree, portability and reliability are related to each other through a common sub-concept of self-containedness.

Similar observations can be made about other types of NFRs than performance, not only concerning the few classification schemes shown here but many other ones not shown here as well, including the one in [19].

Not surprisingly, NFRs play a critical role in the area of architectural design, and a classification scheme is provided in the context of ATAM evaluations [20], e.g., distinction of runtime qualities (availability, performance, security) from development time qualities (modifiability, integration). ATAM also addresses risk, while considering a hierarchy of architecture, process and organization.

Now, what should a software practitioner do then, when even well-known classification schemes are inconsistent with one another, not only terminologically but also categorically?

A software practitioner should be aware of some of the well known classification schemes, such as ISO 9126 - an international standard for the evaluation of software quality, and consider one or more to adopt with some tailoring. No matter what classification scheme a software practitioner might choose to adopt, the most important thing to bear in mind is that s/he should know what s/he means by an NFR term, such as performance, so that the meaning of such an NFR can be communicated with the user as well as with system/software developers so that the end product will behave as expected.

4 Representations of Non-Functional Requirements

Requirements dealing with NFRs are usually separated from the functionality requirements. A usual way to represent them is by means of requirements sentences, which are listed separately under different sections of the technical requirements section. The IEEE standard 830 - Recommended Practice for Software Requirements Specifications - is a good example, where Section 3.2 is used for specifying functional requirements, while the rest of Section 3 is used to describe different types of NFRs.

Some authors propose a structure around the requirements sentences as the one proposed by [21] that is comprised of: identification number, NFR type, use case related to it, description, rationale, originator, fit criterion, customer satisfaction, customer dissatisfaction, priority, conflicts, supporting material, and history. All of these are informal, textual information.

In the classic work with SADT [22], a requirements definition should answer three types of questions: why a system is needed (context analysis), what system features will serve to satisfy this context (system functional requirements), and how the system is to be constructed (system non-functional requirements). Although there is no explicit reference to functionality oriented requirements versus quality requirements, SADT's actigrams can be used to indirectly address some of the NFR concerns. An actigram can be associated with four types of information that interact with the activity: input, control, output and mechanism. The spirit of the control information is much related to the idea of non-functionality, since the control arrow in SADT has the purpose to constrain how the activity is performed. As such, SADT provides a way to address quality attributes or constraints.

NFRs are also commonly represented by trees, expressing the concept of NFR clustering or decomposition [4] and also by lists as well.

Some authors have used NFRs in conjunction with a more structured requirements representation notation, e.g., the combination of NFRs with use cases or misuse cases (for example, see [23], [24],[25], [26] and [27]).

NFRs are also represented as restrictions over different parts of a scenario, along with time and location as the contextual information in the scenario description [28]. Here, the representation of a scenario is comprised of: title, goal, context, resources, actors, episodes, exceptions and the attribute constraint, which applies to context,

resources and episodes. The entity context is further divided into geographical location, time and pre-condition.

Among many proposals, however, the goal oriented approaches, as in [2] [3], were the first to treat NFRs in more depth.

In KAOS [2], which perhaps pioneered in promoting goal-oriented requirements engineering at least from a functional goal perspective, goals are: "… modelled by intrinsic features such as their type and attributes, and by their links to other goals and to other elements of a requirements model." KAOS addresses both functional goals and non-functional goals, which are formalized in terms of operators, such as Achieve, Maintain and Avoid, and by activities based on temporal logic augmented with some special temporal operators. For instance, KAOS offers special operators for concepts, such as "sometime in the future" and "always in the future unless". Thanks to their formal nature, representations in KAOS are amenable to automatic verification and reasoning. In KAOS, goals are characterized as a set of high level constraints over states. For instance, an informal goal: {{Goal Maintain [DoorsClosedWhileMoving]}} [2] can be expressed by the following formula:

$\{\{\forall\ tr: Train, loc, loc': Location\ At(tr, loc) \wedge o\ At(tr, loc') \wedge loc <> loc' \Rightarrow tr.Doors = 'closed' \wedge o(tr.Doors = 'closed')\}\}$.

The above formula, where "o" is a temporal operator denoting next state, means that while a train moves from one location to another location, its doors much be closed during the move.

Although the KAOS' representation language does not differentiate between functional and non-functional goals, the KAOS graphical AND/OR graph makes the differentiation. Goals that can be assigned to individual agents need no further decomposition and can be "operationalized", that is, they can be described in terms of constraints.

Not unlike KAOS, the NFR Framework also promotes goal orientation, but with the main emphasis on NFRs. In the NFR Framework, non-functional requirements are treated as softgoals, i.e., goals that need to be addressed not absolutely but in a good-enough sense. This is in recognition of the difficulties that are associated with both the problem and the corresponding solution. Concerning the problem statement, it is often times extremely difficult, if not impossible, to define an NFR term completely unambiguously without using any other NFR term, which in turn will have to be defined. Concerning the solution, it is also often times extremely difficult to explore a complete list of possible solutions and choose the best, or optimal, solution, due to various resource limitations such as the time, manpower and money available for such an exploration.

Reflecting the sense of "good enough", the NFR Framework introduces the notion of satisficing, and, with this notion, a softgoal is said to satisfice (instead of satisfy) another softgoal. The NFR Framework offers several different types of contributions whereby a softgoal satisfices, or denies, another softgoal - MAKE, HELP, HURT and BREAK are the prominent ones, as well as AND and OR (these also incorporate the notion of "good enough" rather than "absolute satisfaction"). MAKE and HELP are used to represent a softgoal positively satisficing another, while BREAK and HURT to represent a softgoal negatively satisficing (or denying) another . While MAKE and

BREAK respectively reflect our level of confidence in one softgoal fully satisficing or denying another, HELP and HURT respectively reflect our level of confidence in one softgoal partially satisficing or denying another.

In the NFR Framework, each softgoal or contribution is associated with a label, indicating the degree to which it is satisficed or denied. A label propogation procedure is offered in the NFR Framework in order to determine the effect of various design decisions, regardless of whether they are system-level or software-level. In addition to a label, each softgoal or contribution can also be associated with a criticality value, such as extremely critical, critical, and non-critical.

When using the NFR Framework, NFRs are posted as softgoals to be addressed or achieved, and an iterative and interleaving process is applied in order to satisfice them, through AND/OR decompositions, operationalizations and argumentations. Throughout the process, a visual representation, SIG (softgoal interdependency graph), is created and maintained which keeps track of softgoals and their interdependencies, along with the impact of various decisions through labels. In this sense, a SIG shows how various (design) decisions are rationalized.

In order to alleviate the difficulties associated with the search for knowledge for dealing with NFRs , the NFR Framework offers methods for capturing knowledge of ways to satisfice NFRs and correlation rules for knowledge of the side effects that such methods induce.

As with goals in KAOS, softgoals in the NFR Framework are associated with, and ultimately achieved, by the actions of agents – human, hardware or software. This is consistent with the spirit of the reference model [29], in which (functional) requirements are satisfied through the collaboration between the software system behavior and environment phenomena that are caused by agents in the environment, although here we are also concerned with softgoals that (functional) requirements are intended to help achieve together with environment phenomena. Note that requirements are part of the solution to some problem in a piece of reality, and the notion of softgoals can be used to represent anything non-functional, be it about the problem domain or the solution.

The i* family: i* [30], Tropos [31] and GRL(Goal Requirements Language) [32] inherited the concept of softgoal from the NFR Framework, aiming at dealing with softgoals, or non-functionality related attributes, as a first class modeling concept.

As mentioned in Section 2, the last definition [12] presented was somewhat different from the rest: an NFR is described as a justification of a design decision and as a constraint on the way in which a required functionality may be realized. This is exactly why the proper identification and representation of NFRs play a key role in software engineering, since software engineering is said to be all about decision-making.

As several authors have pointed out, NFRs do need to be transformed through some means, methods or operations. This is also why a goal-oriented representation is so well suited for NFRs - they are initially expressed in general terms as more abstract requirements, but then gradually are further refined into more concrete terms and details.

When NFRs eventually are operationalized, in terms of software operations, entities or constraints, they become the justification for why such operationalizations exist in the software system, i.e., to serve the quality attributes specified as NFR softgoals. If the software engineers are careful enough to maintain the history of the

software construction, they will then be able to explain and justify why such operationalizations exist. It goes without saying that this argument is also valid for functionality as well, but the key point here is that the choice on a specific operation to reify a quality concern affects how the overall functionality is achieved. Put differently, different sorts of design decisions are made throughout a software development process, and NFRs act as the criteria for such design decisions (See [33] for a discussion about these design decisions in the context of use cases). As argued in [12], an NFR is not just an after-the-fact justification, but constrains how functionality is realized.

So, if the software engineer uses quality attributes up front during the software development process, there will be a network of explanations, bounded by the usage of rationality, linking the results of decisions with the quality attributes. Work on design rationale [34], drawing on earlier work on the Ibis idea [35], focuses on the justification of decisions, but without taking into consideration pre-existing factors that lead to the decision. Work on design rationale can benefit from better ways of dealing with the dynamic and contextual nature of software design and with the limitations of working with the myriad of possible alternatives and their justifications [36].

Integration of functionality and other qualities is said to be essential. Although no one would dispute that truism, few have proposed or advocated a process that really intertwines these two classes of requirements. Goal-oriented methods, such as the NFR Framework, KAOS and the i* family, are the few exceptions that make the consideration of non-functionality as a first class concept, being intertwined with the functionality, as they are reified. However, it shouldn't come as a surprise that each approach has its own particular ways of doing this interweaving of functionality and quality attributes, with several distinct variations and without necessarily keeping the development history.

So what? In a nutshell, the point we are trying to make is twofold: not only non-functional requirements need to be stated up front, but they can help the software engineer make design decisions, while also justifying such decisions. However, in order to take this into consideration, it is necessary that quality attributes not be considered just as a separate set of requirements, but with the consideration of the functionality throughout the development process.

The NFR Framework and the i* family have an intrinsic characteristic that sets them apart from other NFR methods - the reliance on a qualitative approach towards NFRs or softgoals. In the heart of the NFR Framework lies the concept that softgoals are idealizations and, as such, without all its defining properties necessarily established. This characteristic is similar to the notion of "bounded rationality" put forward by Herbert Simon [37] when explaining his understanding of the process designers use to make a decision given incomplete information. This qualitative characteristic is built on the ideas of contribution and correlation among softgoals as explained earlier. By means of contributions, a softgoal may be decomposed up to the level of operationalization, and, by correlations, different softgoals may interfere among themselves. A network of such contributions and correlations makes it possible to carry out different sorts of qualitative reasoning. The semantics of such a network is given by relationships over three dimensions a) decompositions over an AND/OR tree, b) contributions among sub-trees and c) correlations among different softgoals, all leading to the formation of a softgoal interdependency graph (SIG).

While the NFR Framework's focus is on NFRs, as described at the end of Section 2, NFRs exist in relation to functional things. UML is a functionally-oriented Object-Oriented analysis and design language, and one of the first works to detail how the NFR Framework could help attain better UML models is described in [38], which presents a process for linking NFR graphs with UML models. The central idea is to qualitatively realize softgoals in the UML models. The realization for UML classes, for instance, was based on introducing attributes to classes, methods, or constraints over the attributes.

In i* [30], the links among softgoals and operations (tasks or resources) are more explicit, since modeling is carried out simultaneously in the context of the Strategic Rationale (SR) diagram. In a SR diagram, means-ends relationships (i* specialization operator) can show how choices (tasks) are related to different softgoals, while also showing the pros and cons of each selection.

The NFR Framework presented in this Section accommodates any classification scheme that was discussed in Section 3, through AND/OR decompositions, and goes beyond by offering those concepts of operationalizations and argumentations, together with positive/negative correlations. In terms of these concepts, the NFR Framework helps rationalize design decisions – both system-level and software-level. For representation of NFRs, it recognizes NFRs as softgoals and relatioships between them as partial/full positive/negative contributions.

5 Future Directions

There have been several pieces of work that further explored the concepts of softgoals, while shaping some scenario for future directions. We classify them in six areas: software variability, requirements analysis, requirements elicitation, requirements reusability, requirements traceability and aspect-oriented development. Each of these areas has explored particular aspects of the idea of a SIG (Softgoal Interdependency Graph).

When dealing with software product lines, the idea of variability is critical, since, at some parts of a product line, architecture variation points will exist to enable the production of different alternatives necessary to compose different products out of a single common architecture. The works of [39] [40] [41] explored the fact that the alternatives are intrinsic in SIG models, since they are AND/OR graphs. Using a goal-oriented approach to product lines brings a seamless way of producing product line architectures, since features are not only identified, but also justified, in terms of softgoals.

One of the major advantages of the qualitative approach of a SIG is that it facilitates analysis. The very idea that there can be different types of relationships among softgoals and between softgoals and operationalizations in a SIG brings the opportunity to conduct analysis, by propagating the impact of decisions along the correlation contribution relationships. Using the concept of label propagation over a SIG graph, it is possible to evaluate how a given operationalization of an NFR will impact the whole graph. The original process was devised in the NFR Framework [3] and variations followed [42] [43]. Using the idea of label propagation, different types of analysis could be performed early on before committing to an architecture or to code, as seen in the exploration of security concerns [44], in visually choosing operationalizations [45], and in casting an i* model analysis as a SAT problem [46].

Eliciting requirements requires use of different sets of techniques, but most of them are centered on discovering functionality only. Work on goal elicitation needs to consider both functionality and other qualities. Work in this regard includes a proposal on an elicitation scheme that departs from an extended lexicon [47], repertory grid techniques to help the clarification of naming during the elicitation process [48], and Personal Construct Theory to elicit contribution among softgoal [49].

The NFR Framework [3] has identified that NFR catalogues, composed of SIG graphs, were an important aspect of building software using the softgoal concept. Later work [50] explored the ideas further on stressing the aspect of reusability, while proposing a method for maintaining a softgoal organization aimed for reuse. The idea of a goal centric traceability based on softgoal graphs is explored in [51] [52], in which the softgoal network is used as a baseline for explaining changes over software evolution.

The relationship among the quality attributes and aspect-oriented development has been explored in [53] and [54]. Later on, others have identified the important role the NFR Framework - in particular, softgoals - plays in dealing with early aspects [27], [55] [56] [57] [58] [59] (See [60] for a survey on the topic).

Although these recent results helped further understanding of the general concept of quality requirements and opened new paths for future research, other issues need further advance as well, such as the integration of NFRs into other requirements models, such as the reference model [29] and the four variable model [61] which have had significant influences in the area of requirements engineering. Although these models address performance or accuracy concerns, they are essentially functional models and without the notion of goals. As briefly mentioned in Section 4, for example, the reference model states that (functional) requirements are satisfied through the collaboration between the functional behavior of the software system and the (functional) phenomena in the environment. KAOS [2] goes beyond these functional models and introduces general types of softgoals for the overall system, while addressing performance, accuracy and security concerns for the software system.

We also agree with [11] on the need for further empirical research on the use of NFRs during requirements engineering and on the usage of ethnographical studies on how software teams deal with quality issues as requirements. We understand that this research should be conducted with real projects, both in lab situations as well as on industry projects, to improve our knowledge on dealing with quality issues early on.

Acknowledgements. We appreciate the comments from Barbara Paech on an earlier draft, which significantly helped us improve the paper in a more understandable manner.

References

1. Mylopoulos, J., Chung, L., Nixon, B.: Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. IEEE Trans. Softw. Eng. 18(6), 483–497 (1992), http://dx.doi.org/10.1109/32.142871
2. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: Proceedings of the 5th IEEE international Symposium on Requirements Engineering, August 27-31, 2001, p. 249. IEEE Computer Society, Washington (2001)

3. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. International Series in Software Engineering, vol. 5, p. 476. Springer, Heidelberg (1999)
4. Barbacci, M., Longstaff, T.H., Klein, M.H., Weinstock, C.B.: Quality Attributes, Technical Report CMU/SEI-95-TR-021, ESC-TR-95-021 (December 1995)
5. IEEE Standard 1061-1992 Standard for a Software Quality Metrics Methodology. Institute of Electrical and Electronics Engineers, New York (1992)
6. Freeman, P.A.: Software Perspectives: The System is the Message. Addison-Wesley, Reading (1987)
7. QFD Institute, Quality Function Deployment, http://www.qfdi.org/
8. Hauser Jr., Clausing, D.: The house of quality. Harvard Business Review 66(3), 63–73 (1988)
9. Yeh, R.T., Zave, P., Conn, A.P., Cole, G.E.: Software Requirements Analysis — New Directions and Perspectives. In: Vick, C.R., Ramamoorthy, C.V. (eds.) Handbook of Software Engineering, Van Nostrand Reinhold Co. (1984)
10. Glinz, M.: On Non-Functional Requirements. In: 15th IEEE International Requirements Engineering Conference (RE 2007), pp. 21–26 (2007)
11. Paech, B., Kerkow, D.: Non-Functional Requirements Engineering - Quality is Essential. In: 10th Anniversary International Workshop on Requirements Engineering: Foundation for Software Quality, REFSQ 2004 (2004), http://www.sse.uni-essen.de/refsq/downloads/toc-refsq04.pdf
12. Landes, D., Studer, R.: The Treatment of Non-Functional Requirements in MIKE. In: Botella, P., Schäfer, W. (eds.) ESEC 1995. LNCS, vol. 989, pp. 294–306. Springer, Heidelberg (1995)
13. A lexical database of English, http://wordnet.princeton.edu/
14. ISO/IEC 9126-1:2001(E): Software Engineering - Product Quality - Part 1: Quality Model (2001)
15. Jureta, I.J., Faulkner, S., Schobbens, P.-Y.: A more expressive softgoal conceptualization for quality requirements analysis. In: Embley, D.W., Olivé, A., Ram, S. (eds.) ER 2006. LNCS, vol. 4215, pp. 281–295. Springer, Heidelberg (2006)
16. Roman, G.-C.: A Taxonomy of Current Issues in Requirements Engineering. IEEE Computer, 14–21 (April 1985)
17. Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., MacLeod, G.J., Merritt, M.J.: Characteristics of Software Quality. North-Holland, Amsterdam (1978)
18. Grady, R., Caswell, D.: Software Metrics: Establishing a Company-wide Program. Prentice-Hall, Englewood Cliffs (1987)
19. Bowen, T.P., Wigle, G.B., Tsai, J.T.: Specification of Software Quality Attributes, Report RADC-TR-85-37, vol. I (Introduction), vol. II (Software Quality Specification Guidebook), vol. III (Software Quality Evaluation Guidebook), Rome Air Development Center, Griffiss Air Force Base, NY (February 1985)
20. Bass, L., Nord, R., Wood, W., Zubrow, D.: Risk Themes Discovered Through Architecture Evaluations, Technical Report CMU/SEI-2006-TR-012, ESC-TR-2006-012 (2006)
21. Robertson, S., Robertson, J.: The Volere requirements process, Mastering the Requirements Process. Addison-Wesley, London (1999)
22. Ross, D.T.: Structured Analysis (SA): A Language for Communicating Ideas. IEEE Trans. Softw. Eng. 3(1), 16–34 (1977), http://dx.doi.org/10.1109/TSE.1977.229900
23. Chung, L., Supakkul, S.: Representing nFRs and fRs: A goal-oriented and use case driven approach. In: Dosch, W., Lee, R.Y., Wu, C. (eds.) SERA 2004. LNCS, vol. 3647, pp. 29–41. Springer, Heidelberg (2006)
24. Herrmann, A., Paech, B.: MOQARE: misuse-oriented quality requirements engineering. Requir. Eng. 13(1), 73–86 (2008)

25. Cysneiros, L.M., do Prado Leite, J.C.: Using UML to reflect non-functional requirements. In: Stewart, D.A., Johnson, J.H. (eds.) Proceedings of the 2001 Conference of the Centre For Advanced Studies on Collaborative Research. IBM Centre for Advanced Studies Conference, vol. 2. IBM Press (2001)
26. Alexander, I.: Misuse cases help to elicit non-functional requirements. Computing & Control Engineering Journal 14(1), 40–45 (2003)
27. de Sousa, T.G.M.C., Castro, J.F.B.: Towards a Goal-Oriented Requirements Methodology Based on the Separation of Concerns Principle. In: Anais do WER 2003 - Workshop em Engenharia de Requisitos, Piracicaba-SP, Brasil, November 27-28, 2003, pp. 223–239 (2003), http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER03/georgia_souza.pdf
28. Leite, J.C., Hadad, G., Doorn, J., Kaplan, G.: A Scenario Construction Process. Requirements Engineering Journal 5(1), 38–61 (2000)
29. Gunter, C., Gunter, E., Jackson, M., Zave, P.: A Reference Model for Requirements and Specifcations. IEEE Software, 37–43 (2000)
30. Yu, E.S.K.: Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the Third IEEE International Symposium on Requirements Engineering, pp. 226–235 (1997)
31. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the Tropos project. Information Systems 27(6), 365–389 (2002)
32. Amyot, D., Mussbacher, G.: URN: Towards a new standard for the visual description of requirements. In: Sherratt, E. (ed.) SAM 2002. LNCS, vol. 2599, pp. 21–37. Springer, Heidelberg (2003)
33. Dutoit, A.H., Paech, B.: Rationale-based use case specification. Requirements engineering 7(1), 1–3 (2002)
34. Potts, C., Bruns, G.: Recording the reasons for design decisions. In: Proceedings of the 10th international Conference on Software Engineering. International Conference on Software Engineering, Singapore, April 11-15, 1988, pp. 418–427. IEEE Computer Society Press, Los Alamitos (1988)
35. Kunz, W., Rittel, H.W.J.: Issues as Elements of Information Systems, Working Paper No. 131 (July 1970); Studiengruppe für Systemforschung, Heidelberg, Germany (reprinted, May 1979)
36. Dutoit, A.H., McCall, R., Mistrík, I., Paech, B. (eds.): Rationale Management in Software Engineering. Springer, Heidelberg (2006)
37. Simon, H.A.: The Sciences of the Artificial, 3rd edn. The MIT Press, Cambridge, MA (1977)
38. Cysneiros, L.M., Leite, J.C.: Nonfunctional Requirements: From Elicitation to Conceptual Models. IEEE Trans. Softw. Eng. 30(5), 328–350 (2004), http://dx.doi.org/10.1109/TSE.2004.10
39. Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E.S.K., Mylopoulos, J.: On Goal-based Variability Acquisition and Analysis. In: RE 2006, pp. 76–85 (2006)
40. Yu, Y., Lapouchnian, A., Liaskos, S., Mylopoulos, J., Leite, J.C.: From goals to high-variability software design. In: An, A., Matwin, S., Raś, Z.W., Ślęzak, D. (eds.) Foundations of Intelligent Systems. LNCS, vol. 4994, pp. 1–16. Springer, Heidelberg (2008)
41. González-Baixauli, B., Laguna, M.A., Leite, J.C.: Using Goal-Models to Analyze Variability. In: First International Workshop on Variability Modelling of Software-Intensive Systems, VaMoS 2007, Proceedings, Limerick, Ireland, January 16-18, 2007, pp. 101–107, Lero Technical Report 2007-01 2007 (2007)
42. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Reasoning with Goal Models. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 167–181. Springer, Heidelberg (2002)

43. Kaiya, H., Horai, H., Saeki, M.: AGORA: Attributed Goal-Oriented Requirements Analysis Method. In: Proceedings of the 10th Anniversary IEEE Joint international Conference on Requirements Engineering, September 09-13, 2002, pp. 13–22. IEEE Computer Society, Washington (2002)
44. Liu, L., Yu, E., Mylopoulos, J.: Security and Privacy Requirements Analysis within a Social Setting. In: Proceedings of the 11th IEEE international Conference on Requirements Engineering, September 08-12, 2003, IEEE Computer Society, Washington (2003)
45. Gonzalez-Baixauli, B., Leite, J.C., Mylopoulos, J.: Visual Variability Analysis for Goal Models. In: Proceedings of the Requirements Engineering Conference, 12th IEEE international, September 06-10, 2004, pp. 198–207. IEEE Computer Society, Washington (2004), http://dx.doi.org/10.1109/RE.2004.56
46. Horkoff, J., Yu, E.S.K.: Qualitative, Interactive, Backward Analysis of i* Models. In: iStar 2008, pp. 43–46 (2008)
47. Oliveira, A.P.A., Leite, J.C., Cysneiros, L.M.: AGFL - Agent Goals from Lexicon - Eliciting Multi-Agent Systems Intentionality. In: iStar 2008, pp. 29–32 (2008)
48. Niu, N., Easterbrook, S.M.: Managing Terminological Interference in Goal Models with Repertory Grid. In: RE 2006, pp. 296–299 (2006)
49. González-Baixauli, B., Leite, J.C., Laguna, M.A.: Eliciting Non-Functional Requirements Interactions Using the Personal Construct Theory. In: RE 2006, pp. 340–341 (2006)
50. Cysneiros, L.M., Werneck, V., Kushniruk, A.: Reusable Knowledge for Satisficing Usability Requirements. In: RE 2005, pp. 463–464 (2005)
51. Cleland-Huang, J., Settimi, R., BenKhadra, O., Berezhanskaya, E., Christina, S.: Goal-centric traceability for managing non-functional requirements. In: Proceedings of the 27th international Conference on Software Engineering, ICSE 2005, St. Louis, MO, USA, May 15-21, 2005, pp. 362–371. ACM, New York (2005), http://doi.acm.org/10.1145/1062455.1062525
52. Cleland-Huang, J., Marrero, W., Berenbach, B.: Goal-Centric Traceability: Using Virtual Plumblines to Maintain Critical Systemic Qualities. IEEE Trans. Softw. Eng. 34(5), 685–699 (2008), http://dx.doi.org/10.1109/TSE.2008.45
53. Grundy, J.C.: Aspect-Oriented Requirements Engineering for Component-Based Software Systems. In: Proceedings of the 4th IEEE international Symposium on Requirements Engineering, RE, June 07-11, 1999, pp. 84–91. IEEE Computer Society, Washington (1999)
54. Moreira, A., Araújo, J., Brito, I.: Crosscutting quality attributes for requirements engineering. In: Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering, SEKE 2002, Ischia, Italy, July 15-19, 2002, vol. 27, pp. 167–174. ACM, New York (2002), http://doi.acm.org/10.1145/568760.568790
55. Yu, Y., Leite, J.C., Mylopoulos, J.: From Goals to Aspects: Discovering Aspects from Requirements Goal Models. In: 12th IEEE international Proceedings of the Requirements Engineering Conference, September 06-10, 2004, pp. 38–47. IEEE Computer Society, Washington (2004), http://dx.doi.org/10.1109/RE.2004.23
56. Brito, I., Moreira, A.: Integrating the NFR framework in a RE model. In: EA-AOSD 2004: Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, held in conjunction with the 3rd International Conference on Aspect-Oriented Software Development, Lancaster, UK, March 22-26 (2004), http://trese.cs.utwente.nl/workshops/early-aspects-2004/Papers/BritoMoreira.pdf
57. Alencar, F., Silva, C., Moreira, A., Araújo, J., Castro, J.: Identifying Candidate Aspects with I-star Approach. In: Early Aspects 2006: Traceability of Aspects in the Early Life Cycle, pp. 4–10 (2006)

58. de Padua Albuquerque Oliveira, A., Cysneiros, L.M., do Prado Leite, J.C., Figueiredo, E.M., Lucena, C.J.: Integrating scenarios, i*, and AspectT in the context of multi-agent systems. In: Proceedings of the 2006 Conference of the Center For Advanced Studies on Collaborative Research, CASCON 2006, Toronto, Ontario, Canada, October 16-19, 2006, p. 16. ACM, New York (2006), http://doi.acm.org/10.1145/1188966.1188988
59. da Silva, L.F., Leite, J.C.: Generating Requirements Views: A Transformation-Driven Approach. ECEASST 3 (2006)
60. Yu, Y., Niu, N., González-Baixauli, B., Mylopoulos, J., Easterbrook, S., Leite, J.C.: Requirements Engineering and Aspects. In: Design Requirements Engineering: A Ten-Year Perspective. Lecture Notes in Business Information Processing, pp. 432–452. Springer, Heidelberg (2009)
61. Parnas, D.L., Madey, J.: Functional Documentation for Computer Systems. Science of Computer Programming 25(1), 41–61 (1995)