Non-functional Requirements as Qualities, with a Spice of Ontology

Feng-Lin Li, Jennifer Horkoff, John Mylopoulos
University of Trento
Trento, Italy
{fenglin.li@, {horkoff, jm}@disi.}unitn.it

Alexander Borgida
Rutgers University
New Brunswick, USA
borgida@cs.rutgers.edu

Renata S. S. Guizzardi, Giancarlo Guizzardi
Federal University of Espírito Santo (UFES)
Vitória, Brazil
{rguizzardi, gguizzardi}@inf.ufes.br

Lin Liu
Tsinghua University
Beijing, China
linliu@tsinghua.edu.cn

***Abstract*—We propose a modeling language for non-functional requirements (NFRs) that views NFRs as requirements over qualities, mapping a software-related domain to a quality space. The language is compositional in that it allows (recursively) complex NFRs to be constructed in several ways. Importantly, the language allows the definition of requirements about the *quality of fulfillment* of other requirements, thus capturing, among others, the essence of probabilistic and fuzzy goals as proposed in the literature. We also offer a *methodology* for systematically refining informal NFRs elicited from stakeholders, resulting in unambiguous, de-idealized, and measurable requirements. The proposal is evaluated with a requirements dataset that includes 370 NFRs crossing 15 projects. The results suggest that our framework can adequately handle and clarify NFRs generated in practice.**

***Index Terms*—Non-functional requirements, goal models, software qualities, ontologies**

I. Introduction

Non-functional requirements (NFRs) — such as usability, maintainability, security and performance — have been difficult to deal with since the very beginning of Requirements Engineering (RE) back in the '70s. NFRs are known to have a make-or-break status in software development projects, but are difficult to treat formally.

In RE research and practice, NFRs have been treated in one of two ways: (a) they included all requirements that were not functional (hence their name); (b) they were requirements on quality of the system-to-be, such as usability, maintainability and the like. The former approach bundles together very different kinds of requirements and makes it hard to come up with any kind of formal treatment. The latter approach has led to standards like ISO/IEC 25010 [1]. However, these standards do not say much about the exact nature of the qualities nor how to exploit them in dealing with NFRs.

One of the proposals that attempted to deal with NFRs in depth was the NFR framework (NFRF), first proposed in 1992 [2] and extended into a monograph [3]. In this proposal, NFRs were modeled as "softgoals" — goals with no clear-cut criterion for success. The NFRF offered a simple representation that allowed basic reasoning, such as: if I make design decisions A, B and C, how am I doing with respect to softgoal SG? However, goals lacking a clear criterion of satisfaction (i.e., softgoals) turn out to be not always NFRs — most early requirements, as elicited from stakeholders are also "soft". For instance, when a stakeholder says "*Upon request, the system shall schedule a meeting*", this is also vague and needs to be made more firm: Do we allow requests for any time (e.g., weekends)? Should the system notify participants about the scheduled meeting? Should it account for contingencies (e.g., power outage)? etc. Our conclusion is that softgoals constitute a useful abstraction for *early* requirements, both functional and non-functional, rather than just non-functional ones.

But this conclusion begs the next question: what then are NFRs, how do we model them and how do we use these models in the RE process? We begin with treating them as "*qualities*", and look to foundational ontologies to tell us precisely what qualities are [4]. Foundational ontologies have been defined in the research area known as Applied Ontology (AO) and they include the most general concepts needed for any domain, such as *object*, *event* and *quality*. Prominent foundational ontologies include DOLCE [5] and UFO [6]. In these ontologies, a quality is defined as an individual (instance) with the power to connect the entity it qualifies (its *subject*) with a *value* in a geometric *quality space*.

NFRs are often specified in idealized and/or vague terms, making it hard to assess their fulfillment. Take, for example, NFRs from the PROMISE dataset [7]:

- *NFR*-1: "*The product shall return* (*file*) *search results in an acceptable time*."
- *NFR*-2: "*Administrator shall be able to activate a prepaid card via the Administration section within 5 sec.*"
- *NFR*-3: "*The website shall be available for use 24 hours per day, 365 days per year.*"
- *NFR*-4: "*The interface shall be appealing to callers and supervisors.*"

These NFRs are problematic for a number of reasons:

- *NFR*-1 is vague, and therefore not measurable.
- It is unclear whether *NFR*-2 is strict or gradable (can be relaxed). E.g., would "*5.7 sec.*" do? If gradable, what are the constraints on the possible values?
- *NFR*-3 is idealized and unsatisfiable as such, given all the contingencies that could render it unfulfilled (e.g., power failures, strikes, government shutdowns, etc.)

Accepted for publication by IEEE. © 2014 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/ republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

- For *NFR*-4, some users may report that the interface is appealing while others do not agree. This is due to the fact that some qualities (e.g., *look*, *appearance*) can be subjective, resting "*in the eye of the beholder*".

The aim of this work is to propose a language for modeling NFRs, addressing the challenges listed above. Our proposal makes the following contributions:

- Adopts an ontological interpretation of NFRs based on qualities in foundational ontologies.
- Offers a compositional modeling language for capturing NFRs, where the subjects of involved qualities can be identified using (arbitrarily nested) notation, resembling feature structures in linguistics [8].
- Identifies three (combinative) meta-qualities for talking about the fulfillment of a requirement: universality of a proposed solution, gradability of the fulfillment of the requirement, and agreement among stakeholders that indeed the given requirement is satisfied.
- Proposes a goal-oriented requirements methodology for *refining* ambiguous, (practically or logically) unsatisfiable, or vague NFRs to unambiguous, de-idealized and measurable ones.

The remainder of the paper is structured as follows. Section II introduces the research baseline for this work, section III and IV present a language for capturing NFRs treated as qualities. Section V proposes a goal-oriented methodology for refining NFRs, while section VI evaluates the proposal using a publicly available dataset of requirements. Section VII reviews and correlates related work, and section VIII concludes and offers suggestions for future work.

II. Research Baseline

Requirements as goals. Goal-Oriented Requirements Engineering (GORE) is founded on the premise that requirements are goals that stakeholders want to fulfill. Key GORE proposals include seminal work on the KAOS project [9], *i** [10], and Techne [11], as well as the above mentioned NFRF [3].

In GORE, goals can be refined to other goals through AND/OR refinement. In this paper, we distinguish between (i) functional goals that need to be achieved by functions performed by the system or an external actor, and (ii) quality goals that capture qualities of the system. Functional goals are operationalized by tasks/functions, while quality goals are operationalized by quality constraints, as in Techne [11]. For example, "*collect traffic info*" is a functional goal *FG*#1 that might be operationalized by tasks "*use fixed sensors*" or "*use mobile phone with GPS*", while "*collected traffic info in real-time*" is a quality goal related to *FG*#1. As the name suggests, operationalization makes requirements operational [12], either by providing a task that fulfills a functional goal, or by offering a formally specified Boolean constraint that measures whether a quality goal is fulfilled.

Qualities as mappings. Ontologically speaking, a ***quality*** is defined as a basic perceivable or measurable characteristic that inheres in and existentially depends on its subject [5][6]. The subject can be an object, process, action/task, goal, as well as collectives of objects, processes, and so on. In proposals such as DOLCE [5] and UFO [6], quality is a particular (i.e., instance), e.g. *cost*#1 represents the cost of a specific trip. Each quality has a ***quality type*** *QT* (e.g., *Cost*), which is associated with a ***quality space*** *QS* (e.g., *EuroValues*). These approaches also differentiate a quality, e.g. *cost*#1, from its value, e.g., 1000€, which is a point or region in the corresponding *QS (EuroValues)*.

Our notion of quality space is based on the notion of *Conceptual Space* put forth by Gardenfors [13]. In this theory, quality spaces should be understood literally, given that these structures are endowed with geometrical and topological properties. For instance, associated with the quality type *Cost* we can have a *EuroValues* space, a one-dimensional structure isomorphic to the positive half-line of 2-place decimal numbers; other quality types such as *Color*, *Security* and *Usability* are associated with multi-dimensional spaces, with proper constraints on the constituting dimensions (reflecting the geometry of the space at hand). This theory can be adapted or extended to address a number of relevant conceptual phenomena, from context-dependent, non-monotonic and analogical reasoning [13] to graded membership in vague regions [14].

In this work, we simplify the rich quality theory by treating a quality *Q* (be ontologically correct, *Q* is a quality type) as a mapping (mathematical function) that takes an individual subject *subj* of type *SubjT*, to a quality value (point or region) in *Q*'s codomain (quality space). For example, as a mapping, the quality "*usability*" takes its subject, say a software system "*the E-movie manager*", to a region "*good*" in its quality space. In RE, qualities are also often applied to entire subject types. E.g. the quality "*processing time*" in *NFR*-1 applies to all possible runs of the system. For a fuller account of our ontological treatment of NFRs, interested readers can refer to [14].

III. Non-functional Requirements as Requirements over Qualities

NFRs as qualities. Adopting a qualities-as-mappings perspective, we model an NFR as a *quality goal* (*QG*) that constrains a quality mapping *Q* to take values in a desired region *QRG* of its quality space for its subject type *SubjT*, and capture a QG using the notation in Eq. 1, which is an abbreviation of $\forall x.\ instanceOf\ (x, SubjT) \rightarrow subregionOf\ (Q(x), QRG)$, meaning that for each individual subject *x* of type *SubjT*, the value of $Q(x)$ should be a sub-region of (including a point in) *QRG*.

$$Q(SubjT) : QRG^{1} \quad (1)$$

In addition, we use ':=' to assign names to expressions, for later reference. E.g., *NFR*-1 can be modeled as QG1 in Example 1, below. *Quality constraints* (*QC*) that operationalize QGs use the same syntax, but must involve measurable regions. E.g., a corresponding quality constraint for QG1 is shown in the same example, as QC1.

Example 1 (*NFR*-1).
QG1:= *processing time* (*file search*): *acceptable*.
QC1:= *processing time* (*file search*): $\leqslant$ *8 sec.*
QC1 is-operationalization-of QG1

[1] If *Q* is an aggregate quality like *universality* and *average*, the argument of *Q* will be a set, whose type is a **power-set,** denoted as $\mathcal{P}(SubjT)$. In this case the syntax will be $Q(\mathcal{P}(SubjT)) : QRG$.

NFRs can be defined over both subject types and individual subjects. In Example 1, the subject "*file search*" is a type, not an individual (in object-oriented terms, a class, not an instance); here we refer to a set of its instantiations, i.e., a set of file searches. The expression of QG1 (QC1) implies a set of QGs (QCs), each of which requires a specific run "*file search #*" to take a processing time value in the acceptable (≤ *8 sec.*) region. Hence QG1 (QC1) is interpreted as "*for each file search, its processing time shall be acceptable* (≤ *8 sec.*)".

Consider another NFR: "*The interface shall be intuitive*". In this case, the subject of the requirement is an individual subject, a singleton: "*understandability* ({*the interface*}): *intuitive*", where "*understandability*" is a quality, "*intuitive*" is the desired quality region in the corresponding quality space where the ease of understanding is relatively intuitive.

Quality domains and codomains. The concept of quality in DOLCE [5] and UFO [6] refers to a broad category of instrinsic properties of entities that can be projected on a quality space (roughly, the basis of a measurement structure that becomes the codomain of the associated quality mapping [15]). Examples can be found in every domain, including color, shape, length, atomic number, electric charge, etc.

For our purposes, we adopt the quality model proposed by the ISO/IEC 25010 standard [1] as our reference. This standard distinguishes two categories of qualities: *qualities in use* and *product qualities*, with five and eight qualities, respectively. Fig. 1 shows the eight product qualities and their refinements. For example, "*usability*" is refined into "*learnability*", "*operability*", "*accessibility*", etc.

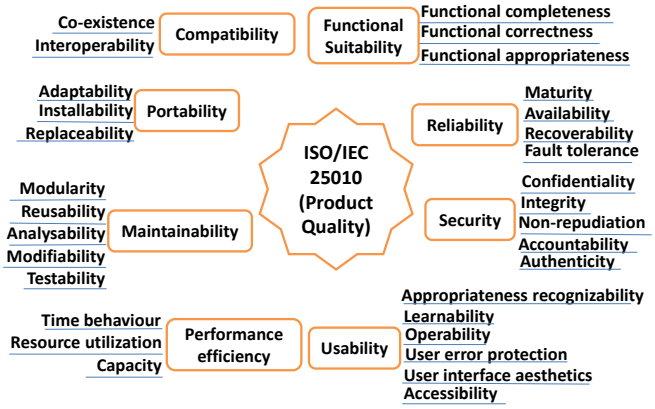

Fig. 1. The eight product qualities in ISO/IEC 25010 (with refinements)

Domains and codomains of qualities are key components in the specification of an NFR. For a specific quality, the set of subject types that it can be applied to constitutes its domain, and the union of all the possible values will form its codomain (quality space).

The domain of a software quality can be any aspect of a software system, including its constituents (code, architecture, requirements, etc.), the software processes that created it, its runtime environment, and the like.

Standards such as ISO/IEC 9126-1 [16] and 25010 [1] are helpful, up to a point, in defining a codomain for qualities. For example, "*availability*" is defined as "*degree to which a system, product or component is operational and accessible when required for use*". Hence it will be associated with a codomain that is a scale ranging from 0% to 100%. We show in Table I possible domains and codomains of 10 frequently used qualities in our evaluation experiment (more details can be found in Section VI).

TABLE I. THE DOMAIN & CODOMAIN OF 10 FREUENTLY USED QUALITIES

Quality	Domain	Codomain
Operability	{*a system*}	{*time to operate*}; {*ease of operating: easy, hard…*}
Availability	{*a system*}	{0% ~ 100%}
Processing / Response time	{*functions/tasks*}	{*time interval*}; {*slow, … fast, …*}
Scalability	{*a system*}	{*simultaneous transactions*}
Learnability	{*a system*}	{*time to learn*}; {*ease of learning*}
Frequency	{*functions/tasks*}	{*numbers per time unit*}
Understandability	{*a system*}	{*ease of understanding*}
Modifiability	{*a system*}	{*time to modify*}
Look and feel	{*a system*}	{*degree of preferences*}

The structure of the codomains of some qualities may be complex, and can differ depending on their subjects. For example, according to ISO/IEC 25010, the codomain of *usability* is a six-dimensional space, with each of its sub-qualities being one dimension. Of course, stakeholders may only be concerned with some of these sub-qualities, in which case a *usability* QG should be refined accordingly. For example, if only *learnability*, *operability*, and *accessibility* are of concern, then the codomain of usability becomes three-dimensional.

By differentiating a quality (type) from the quality spaces it can be projected on, we can account for the possibility of having multiple quality spaces (with measurement structures derived from them) for the same quality (type) [15]. Thus such a quality (type) could map an individual subject to different quality values in their respective quality spaces. For example, as shown in Table I, "*learnability*" can map "*a system*" to either "*easy*/*good*" or "*x minutes of training*" in different quality spaces.

It is important to highlight that our qualities cannot be equated with attributes in the tradition of conceptual modeling [6]. In general, attributes are conventional ascriptions of property values to individuals. Qualities, in contrast, inhere in their bearers, i.e., there is something intrinsic (in the bearers) that makes true a certain property ascription to these bearers. That is, attributes are properties assigned to objects while qualities are properties intrinsic to them [17] (ones we have to design for a system). E.g., "*release date*" and "*serial number*" are merely conventional attributions of certain values to a software system. In contrast, when we state that a system has 50K LOCs or high reliability, there is something *in the system* that makes these statements true.

IV. REPRESENTING COMPLEX NFRS

The preliminary syntax introduced so far, along with a catalogue of qualities allows us to express simple NFRs such as *NFR-1*, but is not sufficient to capture the following aspects of an NFR: (1) a subject restricted by qualifiers, acting as relative clauses, e.g., going from "*activate a pre-paid card within 5*

sec." *to* "*activate a prepaid card <by Administrator> <via the Administration section> within 5 sec.*" (*NFR*-2); (2) NFRs that are unsatisfiable because of blanket use of universal quantifiers (*NFR*-3, but also *NFR*-2); (3) hard constrained NFRs that leave no room for flexibility (e.g., *NFR-2*); and (4) subjective NFRs whose satisfaction depends on the eye of the beholder (e.g., "*appealing look*" in *NFR*-4). To address these issues, we need to enrich our language with new constructs.

A. Qualified Subjects

We extend the basic syntax introduced earlier by allowing its subject type *SubjT* to be restricted by *qualifiers* that consist of <*attribute*: *filler*> pairs referring to *SubjT* (or fillers, when nested)[2]. By using this language, we are able to define particular sets of individual subjects, over which we can talk about concerned qualities. For example, the subject of *NFR*-2, "*activate pre-paid card*", is a software function and can be qualified by the attributes "*actor*" and "*means*", as in Example 2. It represents the set of activations performed by administrators through the admin section (past or future).

Example 2 (*NFR*-2).
activate p-card' :=
activate pre-paid card <***actor***: *Administrator*>
<***means***: *via the Administration section* >.
QG2 := *processing time* (*activate p-card'*): *within 5 sec.*

B. Qualities of Fulfillment

Many requirements can be represented as logical assertions of the form $\forall x\ P(x)$, as in "*For every request* ($\forall x$) *a meeting shall be scheduled* ($P(x)$)" (FR) and "*Every file search* ($\forall x$) *will be completed within 5 sec* ($P(x)$)" (NFR). Inspired by knowledge representation techniques for uncertainty [18], we propose three meta-qualities on the fulfillment of a requirement: (1) *universality*: the degree to which the set of all x satisfies P; (2) *gradability*: the degree to which P holds for each x; (3) *agreement*: the degree to which observers agree P holds for each x. We accordingly define three meta-quality functions, ***U*** for universality, ***G*** for gradability, ***A*** for agreement, that can be applied to requirements, functional or non-functional, to define quality goals.

Universality. The **U** operator aims at limiting universality for its requirement subjects, in that a requirement need not be fulfilled in all cases, but rather in a percentage thereof. E.g., *NFR*-3 can be relaxed as "*the website shall be available 99.5% of the time per year*", expressed as

theWebsite' := *theWebsite*
<***at***: *time units* <***in-period***: *a year*>>
QG3 := *availability*(*theWebsite'*): *100*% //the entire unit
QG3-1 := ***U*** (*QG3*): *99.5*% //99.5% of the units in a year

U takes as argument a set of requirement subject instances, which is of type **power-set**(*SubjT*) (i.e., $\mathcal{P}(SubjT)$), and returns a percentage of the instances for which the requirement is to-be-fulfilled in the linear space 0% ~ 100%. In this case, the subject "*theWebsite'*" has *N* instances representing the system during each unit in a one-year period, and QG3 accordingly has *N* QG instances, with each of them requiring the website to be 100% available for its corresponding time unit. Originally, all QG3's instances are required to hold. It is now relaxed to QG3-1, saying only 99.5% of them need to be satisfied. *NFR*-2 can be relaxed in a similar way, saying *k*% of the activations shall be within 5 seconds.

The **U** operator regulates/modifies the fulfillment of a requirement, either non-functional or functional, from a universal or statistical perspective. For instance, the requirement "*all users shall be authenticated*" can be represented as a functional goal FG that calls for a function *authenticateUser*. If stakeholders can tolerate some failures for this requirement, say 1%, this can be captured by the universality requirement "***U*** (*authenticateUser*): *99%*". During elicitation, it is useful to ask a stakeholder who calls for a universal requirement in the form of "$\forall x\ P(x)$", whether he/she really means it for all x: "Could you live with less, and if so, how much less?". It is also helpful to remind the stakeholder that universal requirements are at the very least harder and more expensive to fulfill, and at worst simply unsatisfiable.

Gradability. The **G** operator allows for partial satisfaction of a requirement. Specifically, **G** maps a requirement to its desired degree of fulfillment on a linear scale 0% ~ 100%.

When evaluating the satisfaction of an NFR that specifies either crisp quality regions such as "≤ 5 *sec.*", "2 ~ 3 *m*" and "100 ~ 200 €", or vague regions like "*fast*", "*high*" and "*low*", the measured or perceived quality value may approach but not be exactly located in the desired region. To accommodate degrees of satisfaction, we use **G** to relax NFRs. For example, *NFR*-2 (captured as QG2) can be relaxed as QG2-1, requiring the processing time value to be *nearly* within the region (0 *sec*., 5 *sec.*], or QG2-2, requiring the processing time to be *90%* in the region. By using **G**, the membership of a time value in the interval (0 *sec*., 5 *sec.*] is made gradable ("*fuzzy*" in the sense of Fuzzy Logic [19]), and we only require a partial membership (e.g., *nearly*, 90%) in that interval for fulfillment. The relaxed membership can be clear or vague; if vague (e.g., *nearly*), it shall eventually be made clear (e.g., *90%*) through operationalization. For details on calculating graded membership based on the theory of quality space, we refer to our companion paper [14].

QG2:= *processing time* (*activate p-card'*): *within 5 sec.*
QG2-1 := ***G*** (*QG2*): *nearly*
QG2-2 := ***G*** (*QG2*): *90%*

G can also be applied to relax NFRs with vague quality regions besides those with crisp regions like (0 *sec.*, 5 *sec.*]. For instance, *NFR*-1 can be relaxed as follows, requiring the processing time value to be *moderately* in the *acceptable* region.

QG1:= *processing time* (*file search*): *acceptable.*
QG1-1:= ***G*** (*QG1*): *moderately.*

The ***G*** operator also captures the degree of fulfillment of functional requirements (FRs). A functional requirement, especially one that calls for multiple/batch tasks, can also be only partially fulfilled. For example, if room equipments have not been returned after a meeting, the scheduling requirement can be seen as *almost* but not *totally* fulfilled.

[2] Our proposed language currently does not provide a built-in set of attributes, which requires an ontology of software systems and of the application domain.

Agreement. Agreement (**A**) is intended to address the subjectivity of qualities. The satisfaction of some NFRs, especially those concerning qualities that depend on human individuals, such as *look*, *attractiveness* and *satisfaction*, is subjective and will vary with the observer who is beholding.

We can make such NFRs objective by operationalization (e.g., "*the interface shall be intuitive*" can be operationalized as "*80% of the new users can operate the system without training*" with the use of **U**), or by using **A** to capture the agreement among observers that a requirement is indeed satisfied. E.g., *NFR*-4 in our list can be rephrased as QG4-1, requiring 80% of the callers and supervisors to agree that QG4 holds.

QG4:= *look* (*the interface*): *appealing*
QG4-1:= ***A*** (*QG4*): *80% of the callers and supervisors*

A relates to the notion of *precision* widely used in Science and Engineering. Precision for a measuring system is defined as the degree of repeatability, i.e., the extent to which multiple measurements lead to the same result. In our case, the measuring system is an observer and a measurement consists of the observer determining whether a requirement is satisfied or not. In this sense, the domain and codomain of **A** consist of a set of requirements, and ratios of observers from a given pool who agree each requirement is satisfied, respectively.

Composition. In practice, a requirement may be specified using multiple applications of the three operators. For example, to make *NFR*-2 practical, we can relax it by using **U**, **G**, or both. As shown below, we first use **G** to relax QG2 as *nearly* being in the region (0 *sec.*, 5 *sec.*], then use **U** to relax the set of all executions of "*activate p-card′*", requiring *95%* of the activation processes to be *nearly* in that interval.

QG2:= *processing time* (*activate p-card′*): *within 5 sec.*
QG2-1:= ***G*** (*QG2*): *nearly*
QG2-3:= ***U*** (*QG2-1*): *95%*

These three operators can be combined in many different ways: **U** over **G** (firstly apply **G** and then **U**, as in the above example), **A** over **G** (e.g., *80% of the users* report the website is *kind of* hard to understand), **G** over **U**/**A** (e.g., *nearly 90%* of the activation takes 5 *sec*., *nearly 80% of the users* report the interface is simple), or even **G** over **U**/**A** over **G** (e.g., *nearly 90%* of the activation takes *nearly* 5 sec.). The full syntax and semantics of proper operator nesting will be part of our future work.

V. A Framework for Goal Models with Qualities

We introduce next a goal modeling framework enriched with qualities and a methodology for refining early and informal NFRs to unambiguous, satisfiable and measureable ones. The framework and methodology is evaluated through the case study on the PROMISE requirements dataset [7] in Section VI.

A. Goal Models with Qualities

Our conceptual model for goal models is shown in Fig. 2 and includes the concepts introduced earlier. In general, we represent a requirement as a Goal, which is further specialized into Functional and Quality Goal. Functional goals are operationalized by functions (i.e., tasks), while quality goals are operationalized by quality constraints. Any goal can be operationalized by a domain assumption. E.g., the functional goal "*Find room for meeting*" may be operationalized by a domain assumption like "*There are enough rooms available for all requested meetings*". Also, by function constraint, a function can be constrained to situations that must hold before/after/during its execution, analogously to pre/post-conditions and invariants. E.g., "*only managers are allowed to activate users*" is a constraint over the function "*activate users*". The three kinds of refinements, namely disambiguation, relaxation and focus, will be introduced in detail in the next section.

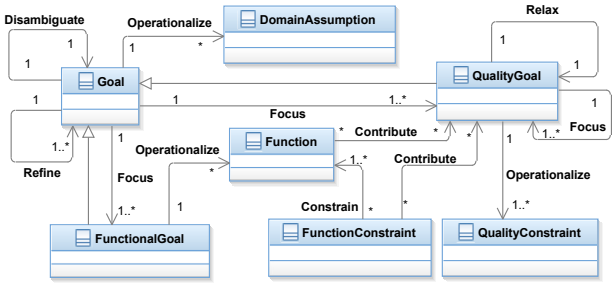

Fig. 2. The conceptual model for the revisited goal modeling framework

B. Building Goal Models during Requirements Analysis

In general, goals elicited from stakeholders are vague, ambiguous, idealized, etc. The aim of requirements analysis is to iteratively refine them into a specification that includes concrete and/or measurable functions, quality constraints and domain assumptions.

Our corresponding methodology is based on iteratively answering the following questions: (1) Is a requirement/goal unambiguous? (2) Is it (practically) satisfiable? (3) How do we make it measurable? In response to these questions, we perform *disambiguation*, *relaxation*, and *focus* refinement, in addition to the usual logical (AND/OR) refinements and operationalization of goal models.

Disambiguation. This is the first phase for capturing and analyzing requirements. On discovering the ambiguity of a given requirement/goal [20], we can keep asking the following questions: (1) What is the subject of the goal? (2) What quality does it refer to (if any)? These questions help identify not only the subject (and the quality) of a goal, but also potential ambiguities. E.g., by focusing on the subject in "*the interface shall have standard menu buttons for navigation*", we find that there are four possible interpretations: (1) there should be buttons all of which should be standard; (2) there should be buttons some of which should be standard; (3) if there are buttons then all of them should be standard; (4) if there are buttons then at least some should be standard. The customer may choose (1), in which case we should refine the goal to "*the user interface shall have menu buttons, and **all** of them shall be standard*". Ambiguity may also arise from word polysemy and multiplicity of structural analyses, interested readers can refer to [20].

Relaxation. After disambiguation, we need to analyze whether a goal is satisfiable or not in practice. As discussed earlier, a requirement may be hard to satisfy because of: (1) the use of universal quantifiers; (2) the specification of categorical quality regions (e.g., "≤ 5 sec."); (3) subjectivity. In

such cases, we use the three operators, **U**, **G**, and **A**, or a combination thereof to relax the requirement to a satisfiable (and acceptable to stakeholders) degree.

Focus. Using focus refinement, we can focus a goal intertwining functionalities with qualities to functional goals and/or quality goals (e.g., "*collect real-time traffic info*" can be stated as a goal, and focused to "*collect traffic info*" and "*timeliness* (*collected traffic info*): *real-time*"), or focus quality goals by concentrating on sub-qualities or parts/elements of the system that are of concern. For quality goals, focusing may move along two dimensions: (1) the quality/sub-quality hierarchy, and (2) the hierarchy of their subjects, generalization or aggregation for some subjects, goal hierarchy for goal subjects. The quality hierarchy we use is defined in the quality standard adopted, such as ISO/IEC 25010 [1].

For example, the quality *usability* has sub-qualities *learnability*, *operability*, *accessibility*, etc., according to ISO/IEC 25010. As shown in Fig. 3 (the meeting scheduler case study, MS), the quality goal concerning "*good usability*" (*MS-QG3*) may be focused into *MS-QG4* and *MS-QG5* that require respectively the system to be easy to learn and operate. Similarly, since a meeting scheduler has different functions, we can further apply *learnability* and *operability* to functionalities such as "*set up a meeting*" and "*reserve a conference room*", obtaining *MS-QG6* and *MS-QG7* respectively.

When applying these two focusing refinements, we should pay attention to the applicability of a quality to a subject. It is not always meaningful to apply a quality to all the parts of its subject, or all the sub-qualities of a quality to its subject. For instance, user friendliness for a system may be focused into user friendliness for its interface, but likely not for its timeslot scheduling component. That is, quality functions are inherently partial w.r.t. a software-related domain.

Note that the two steps, *relaxation* and *focus*, may be interleaved in practice. For instance, the goal G := "*monitor events*" may first be focused into G':= "*monitor suspicious events*", and then relaxed into **U**(*G'*): *98%*".

Operationalization. Once quality and subject have been refined to a suitable granularity, we are left with the problem of operationalizing vague quality values, i.e., making them measurable. E.g., if we have a quality goal such as *MS-QG7* in Fig 3, how do we measure whether it is **easy** to learn?

To answer this question, we need to choose one or more quality dimensions (called metrics in ISO/IEC standards) to measure *learnability*. The ISO/IEC 9126-2 standard [21] can help in this respect, e.g., *learning time*, *help frequency*, etc. This means that operationalization may use several dimensions of the quality space of *learnability*.

Let us assume that *learnability* is measured by *learning time*. To get a reference value for *easy*, a typical way is to determine a comparison class, a set of similar meeting scheduler systems, and apply the quality *learning time* to the set of subjects to get a set of typical quality values. We can then get a reference value from these values, say, the average.

When determining typical time values, we need to focus to the exact subject that the quality being considered inheres in, because typicality may be manifested differently as the subject varies, resulting in values in different regions of a quality space [22]. E.g., typical values for the *easy* learnability region concerning the function *scheduleTimeslot* may be different from those for the function *informParticipants*.

Contribution. In our proposed framework, goals can be focused to functional goals leading to functions, or to quality goals resulting in quality constraints. However, our models do not allow refining a functional goal into a quality goal and vice versa. To address situations where functional elements contribute to the satisfaction of quality goals, we use contribution relationships (*help*, *hurt*, *make*, and *break*) of functions or constraints over functions on relevant quality goals. For instance, the function "*authenticateUsers*" would help the authenticity of a system. Contribution links constitute an important element of tradeoff analysis during RE processes [23] and will be explored in future work.

VI. Evaluation

The PROMISE (PRedictOr Models in Software Engineering) dataset consists of 625 requirements collected from 15 software development projects [7]. Among them, 255 items are marked as functional requirements (FRs) and the remaining 370 non-functional requirements items are classified into 11 categories, such as *Security*, *Performance* and *Usability*. Classification counts are shown in the second column of Table II.

In this section, we describe a comprehensive case study on the 370 PROMISE NFRs. Our aim is twofold: *a*) to evaluate the need for our framework by examining the nature of NFRs in practice; and *b*) to evaluate the expressiveness of our framework by applying it to the set of NFRs of *meeting scheduler*, one of the fifteen projects in the PROMISE dataset. To evaluate *a*), we observe the occurrence of elements in our conceptual model, and evaluate the implicit use of and need for our proposed meta-qualities. To evaluate *b*), we rewrite the set of NFRs of *meeting scheduler* by using our proposed syntax, applying our methodology as described in Section V.

A. The Necesity of our Framework: PROMISE NFRs

We first classified the 370 NFRs according to our ontological classification of requirements. Our classification includes three basic categories of requirements, "functional requirement (FR)", "quality requirement (QR)", and "constraints over function (CF)". These would be modeled by functional goals, quality goals and function constraints in our conceptual model (see Fig. 2), respectively.

Our classification is shown in Table II. Among the 370 items, we identified 187 QRs, 52 FRs, 50 CFs, 61 requirement items that constitute a combination of FRs/CFs and QRs (FR/CF+QR), 12 FRs with constraints over functions (FR+CF), and 8 domain assumptions (DA). Statistically, QRs by themselves account for 51% of the NFRs, and quality-related requirements (QR, FR/CF+QR) account for 67%. Moreover, there are 21 FRs and 36 CFs which we judge to contribute to QRs (e.g., security-related CFs contribute to security), bringing up the total of QR-related requirements to 82% in the sample dataset. These statistics support the claim that most NFRs are indeed quality-related [24], and support the need for an explicit classification of function constraints (CFs).

TABLE II. STATISTICS OF THE CLASSIFICATION OF THE 370 NFRS

NFR Category	Org.	QR	FR/CF + QR	FR	CF	FR +CF	DA
Usability	67	**47**	13+1	5(3)	1(1)	0	0
Security	66	2	11+3	**14(11)**	**32(32)**	4	0
Operational	62	11	10+2	**14**	12(3)	**6**	**7**
Performance	54	**44**	4+1	3(2)	1	1	0
Look and Feel	38	**20**	7+2	9(1)	0	0	0
Availability	21	**21**	0	0	0	0	0
Scalability	21	**19**	0	1	0	1	0
Maintainability	17	8	5	0	4	0	0
Legal	13	11	0	2(2)	0	0	0
Fault tolerance	10	4	2	4(2)	0	0	0
Portability	1	0	0	0	0	0	1
Total	370	187	61	52(21)	50(36)	12	8

Org.: original categorization; QR: quality requirements; FR: functional requirements; CF: constraints over functions; FR + CF: the combination of FR and CF; FR/CF+QR: the combination of FR and QR, or CF and QR; DA: domain assumptions. Interested readers can find the original data of our evaluation at http://goo.gl/8ALJDq.

Examining the data more closely, we find that 151 of the 187 QRs (81%) are classified under usability, performance, availability, look and feel, and scalability; 28 out of 52 FRs (54%) are classified under security and operational requirements; and 32 out of the 50 CFs (64%) are security-related. Moreover, our analysis indicates that the majority of security requirements are, in fact, functional or constraints over functions. E.g., "*The website shall prevent the input of malicious data*", originally labeled as a security NFR, should actually be a functional requirement. Our dataset includes many requirements of the form "*only users with <role> are allowed to perform <action> or access <asset>*". In our classification, these were treated as constraints over functions (CFs), not NFRs, since the system-to-be is required to check whether an actor is authorized to act on an asset.

We analyzed the 248 quality-related NFRs (187 QRs and 61 FR/CF+QRs), and identified 67 unique qualities with 327 occurrences (i.e., 327 QGs). The most frequent ones are operability, availability, processing/response time, and scalability.

TABLE III. THE STATISTICS OF SATISFACTION TYPES AND IMPLICIT PRESENCE OF THE OPERATORS ON THE PROMISE NFRS

Satisfaction Type	NFRs#		Operator Stats.	NFRs#
Ambiguous	5		Universality (**U**)	50
Unsatisfiable	86		Gradability (**G**)	10
Vague	143		Agreement (**A**)	16
Measurable	333			

Our analysis of the 370 NFRs resulted in 481 requirements statements using our framework. These were further classified, as shown in the first two columns of Table III. Note that a statement can be tagged with more than one type, e.g., "*all users shall be authenticated*" is practically unsatisfiable, but also measurable, thus the sum of this classification is greater than 481 (in fact, 567). This classification found 15% (86/567) of the statements to be practically unsatisfiable, 25% (143/567) vague and only 59% (333/567) measurable.

We analyzed the implicit application our three operators **U**, **G**, and **A** to the 481 requirements statements. E.g., "*80% of the users report the user interface is simple*" captures agreement. We show these counts in the last two columns of Table III: 50 **U** (i.e., stating percentages), 10 **G**, and 16 **A**. Our analysis shows that few of the statements have (implicitly) used **U** and **A**. Particularly, **G** is rarely used. Meanwhile, we found that many NFRs, which need to be relaxed to become satisfiable and measurable, have not been adequately dealt with.

Among the 481 statements, 86 of them implicitly or explicitly use universal quantifiers, e.g., *all*, *any* and *each*, (counted as unsatisfiable in Table III) and likely need to be relaxed using **U** to be properly treated. Also, 36 subjective statements are identified, e.g., *look*, *readability*, *usefulness*, etc., indicating that at least another 20 requirements should be relaxed using **A**. Lastly, **G** could be applied to unsatisfiable (e.g., *almost* all), vague or measurable requirements (see the discussion of **G** in Section IV.B), thus all the 476 items (except the 5 ambiguous ones) are candidates for the **G** operator, but only 10 actually (implicitly) used it (e.g., *fast enough*). E.g., "*the interface shall be appealing*", as found in the dataset, is clearly gradable.

B. Using our Proposed Methodology: Meeting Scheduling

We use *meeting scheduler*, one of fifteen projects in the PROMISE requirements dataset, to evaluate the expressiveness of our framework and illustrate how our goal modeling language and methodology can be applied to a realistic case study. Functionally, the *meeting scheduler* is required to create meetings, send invitations, book conference rooms, book room equipment and so on. This example includes 47 NFRs, covering different aspects of the system, such as interoperability, usability, security, user friendliness, etc.

We analyzed the 47 NFRs, and identified 21 QRs, 9 FRs, 14 FR+QR, 2 CF+QR, and 1 DA. We captured the 37 items (excluding the 1 DA and 9 FRs) using our framework, resulting in 58 quality goals (an NFR may refer to more than one quality), concerning 27 unique qualities. Frequently used qualities include interoperability, operability, scalability (concurrent capacity), etc. We managed to rewrite all the 58 QGs using our syntax, validating the expressiveness of our framework. In this project, we did not find ambiguous NFRs.

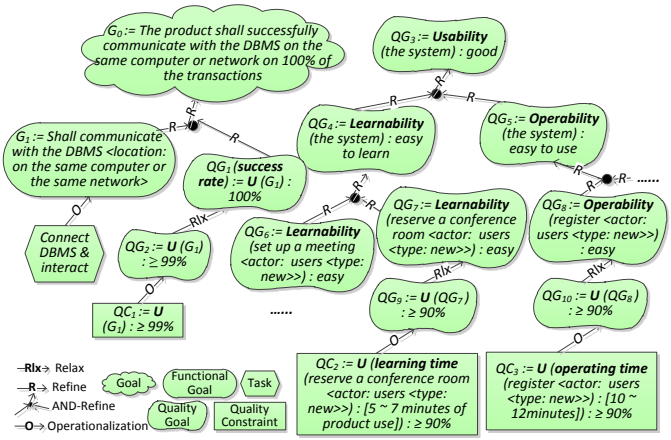

Fig. 3. The NFR model of meeting scheduler (partial)

For space reasons, we only show part of the goal model with a subset of the goal modeling elements in Fig. 3 (note that *focus* is a sub-type of *refine*, and we do not distinguish between them in the model). The full model can be found online:

http://goo.gl/AxNjPf (it has more than 58 QGs because it also includes their refinements). In our model, quality goals are formally captured using our framework rather than informal descriptions stated in natural language. Moreover, these were systematically refined and operationalized into quality constraints by following our methodology, making the informal quality goals satisfiable, and measurable.

C. Summary

We have evaluated the need for our framework using real data. The statistical analysis tells us that quality plays a key role among NFRs in RE practice, and that many requirements identified as NFRs are actually constraints over FRs. These results help to justify our classifications of requirements (Fig. 2). Moreover, many NFRs are ambiguous, (practically) unsatisfiable, vague, and subjective, demonstrating the need for the meta-qualities as introduced in our language. Lastly, we have tested the expressiveness of our framework by capturing the NFRs of *meeting scheduler* using our syntax, and illustrated our methodology by performing an in-depth case study. Our extensive results show that the framework is adequate for covering NFRs in practice, and, in fact, could improve RE practice by prompting for the elicitation of unambiguous, satisfiable and measurable NFRs.

VII. Related Work

A. Definitions of NFRs

On the key topic of what are NFRs and how to deal with them, there have been many efforts. Notably, there have been two recent reviews by Martin Glinz [25] and Chung et al. [26].

Glinz [25] surveys thirteen NFR definitions and suggests his own, based on attributes and constraints. However, his definition focuses on only system NFRs, and does not take into consideration project and process requirements (e.g., development, deployment, maintenance, etc.). In our framework, such processes will serve as subjects, allowing us to specify different kinds of desired qualities of them. Moreover, his proposal does not offer methodological guidance for designing language and/or method for capturing NFRs.

After analyzing a list of NFR definitions, Chung et al. [26] define functional requirements as mathematical functions of the form $f: I \rightarrow O$ (I and O represent the input and output of f), and NFRs are anything that concern characteristics of f, I, O or relationships between I and O. This definition, accordingly, will treat constraints over functions (CFs) as NFRs, e.g., "*updates of the database can only be performed by managers*" is a function constraint in our framework (it requires the system to check who is updating, and is not about the quality of updating), but an NFR according to Chung et al. [26]'s definition.

B. The Treatment of NFRs

General approaches. In general, NFRs are classified into sub-categories and represented by plain or structured sentences. E.g., the IEEE standard 830 [27] classifies NFRs into interface requirements, performance requirements, attributes and design constraints, and document them in a separate section from FRs using plain English. The Volere template [28] presents a structure for requirements sentences, including ID, type, description, rationale, dependencies, etc., all of which are informal and textual information. Moreover, such approaches classify requirements strictly as function/non-functional, and do not support capturing requirements which combine functional and quality aspects. Our evaluation, as well as the work of Svensson et al. [29], has found many such examples in practice. These approaches also do not reflect the cross-cutting concern of NFRs (i.e., a quality can have multiple parts of a system as its subject).

NFRs have been also used along with structural requirements representation languages like UML use case and class diagram, which are widely used to capture FRs in industry [30] [31]. These approaches closely relate NFRs with FRs, associating NFRs with use cases or mapping the operationalizations of NFRs to operations/attributes of UML classes. By relying on the NFR framework [3] to refine and operationalize NFRs, they are able to capture the gradability, but do not take into consideration the universality and agreement of NFRs.

Goal-oriented approaches. Goal-oriented approaches are the first to treat NFRs in depth among many approaches [26]. The NFR framework [2][3] was the first to use vague *softgoals* to capture NFRs. In line with this position, many efforts have been devoted to goal models, resulting in many frameworks, such as *i** [10], Tropos [32], and Techne [11].

The NFR framework [3] has used *type* and *topic*, which are similar to *quality* and *subject*, to represent softgoals (e.g. "*accuracy* [*account*]", wherein "*accuracy*" is a type and "*account*" is a topic). However, it did not further explore qualities (types), subjects (topics) and quality values or spaces, which could give rise to the universality, gradability and agreement of NFRs. Similarly, Jureta et al. [33] have used DOLCE quality to distinguish between NFRs and FRs, and define *softgoal* and *quality constraint*, but they did not focus on the ontological meaning of quality, i.e., what kind of subjects a quality can inhere in within a software system, what is the structure of its value space, what kind of quality value it has (vague vs. crisp), etc.

Earlier work by Jureta et al. [34] has identified similar issues in treating NFRs as softgoals (e.g., subjectivity, imprecision, idealism). They deal with these issues by precisely defining softgoals using templates, emphasizing quantification and eventual formalization of refinements. Softgoals in their approach are de-idealized by finding specific targets derived from benchmarks, while our framework offers a richer set of operators for relaxing quality goals. Unlike our work, their proposal has not yet been validated through application to real data.

Note that we are not the first to use the concept "*quality goal*". Lamsweerde [35] has already proposed this term, but simply treated it as a softgoal.

Quality-oriented approaches. Quality is the most popular term adopted to specify NFRs. Many efforts have been made towards classifying and quantifying qualities, resulting in fruitful quality models and techniques. The famous models include Boehm et al. [36], ISO/IEC 9126-1 [16], ISO/IEC 25010 [1], etc., in which qualities and their interdependencies are usually organized in a hierarchical structure.

One issue with these quality models is that they, even the well-known ones, are neither terminologically nor categorically

consistent with each other [26]. E.g., "*understandability*" is a sub-quality of "*usability*" in ISO/IEC 9126-1 [16], but is a sub-class of "*maintainability*" in Boehm et al. [36]. In our proposal, the categorization of qualities (or NFRs) will be transformed into two open questions: what kind of subjects needs to exist? (e.g., artifacts, functions or processes) and what kind of qualities will inhere in them? Our answer to them is to develop ontologies containing a number of upper-level categories showing what the qualities and subjects can be and let stakeholders to decide depending on the system domain.

Quality quantification is another important aspect on dealing with quality requirements. It is similar to our focus refinement and operationalization: a quality is often decomposed to several sub-qualities and then quantified using metrics [21][38]. One can refer to the example of "*learnability*" adopted from ISO/ IEC 9126-2 [21] and discussed in section V.B.

The planning language (i.e., Planguage), proposed by Tom Glib [38] and widely used in industry, is a typical example along this line: it uses a set of keywords such as *scale*, *meter must* and *plan*, and a *syntax* that captures *fuzzy concepts*, *quantifiers* and *collections* to express and quantify quality requirements. However, Planguage is informal and textual, and does not allow compositional notions for specifying the subjects of concerned qualities as well as the degree of fulfillment of other requirements. Moreover, it does not offer a methodology for refining informal stakeholder goals to unambiguous, satisfiable and measurable requirements, only providing a language to capture the results of such a process.

Uncertainty and vagueness. These two characteristics of requirements have been actively discussed in RE [19][39][40].

Statistical uncertainty has been extensively studied in goal models, e.g., KAOS [39] and Tropos [41]. These approaches use probability theory to propagate the satisfaction of goals in an AND/OR refined hierarchy. Fuzzy logic has been applied to capture the vagueness of requirements for trade-off analysis at early stage [42]. Recently, the vagueness of NFRs has been explored for adaptation [19][40].

Although probability theory and fuzzy logic have been adopted to handle uncertainty and vagueness, they are often not well distinguished: *probabilistic*, which indicates the probability that a requirement can be true or false, is usually confused with *fuzzy*, which implies the degree to which a requirement can be satisfied.

Our proposal captures these two features by using the **U** and **G** operators. Moreover, our framework allows the combination of these features, which is very practical: either fuzzy probability (e.g., *nearly 90% of the time*) or probability over fuzzy (e.g., *nearly 5 sec. 90% of the time*).

Summary. We summarize the related work with regard to the three key features of NFRs, **U**, **G**, and **A**, in Table IV, wherein '√' and '×' means supported and unsupported, respectively. One should distinguish between vague and gradable: vague requirements are requirements with indeterminate satisfaction conditions (e.g., *high* availability, *low* cost, and *fast* response) while gradable means the degree of fulfillment of a requirement can be graded, resembling the fuzzy membership in Fuzzy Logic. Please also note that although agreement has been recognized as an important aspect in RE [43] since the early '90s, few of the surveyed approaches have captured it.

TABLE IV. AN OVEWVIEW ON COMPARING RELATED APPRAOCHES

App.	Source	Technique	T	U	G	A	C
General	IEEE 830 [27]	English	N	√	√	√	√
	Robertson et al.[28]	Volere	N*	√	√	√	√
	Supakkul et al.[30]	Use case	S	×	×	×	×
	Cysneiros et al.[31]	Class diag.	S	×	×	×	×
	Whittle et al. [40]	Fuzzy logic	F	×	√	×	×
	Yen et al. [42]	Fuzzy logic	F	×	√	×	×
Goal Oriented (GORE)	NFR framework[3]	Goal model	S	×	√	×	×
	KAOS [9][39]	Probability	F	√	×	×	×
	*i** [10]	*i**	S	×	√	×	×
	Tropos [32][41]	Probability; Fuzzy logic	F	√	√	×	×
	Techne [11][33]	Techne	S	×	√	×	×
	Jureta et al. [34]	Goal model	S	×	√	×	×
Quality Oriented	Tom Glib [38]	Planguage	N*	√	√	×	×
GORE+ Quality	**Our Work**	**Ontology**	**F**	√	√	√	√

App.: approach; T: type; N: informal (plain English); N*: informal (structured English); S: semi-formal; F: formal; U: universality; G: gradability; A: agreement; C: compositional

There have been some discussions about universality (**U**). Berry et al. [44] argues that the use of "*all*" in requirements specification documents expressed in natural language is "dangerous" and practically unfulfillable when used to define domain assumptions, but a "laudable goal" when used to describe requirements. We are arguing the opposite in our work. Use of universals in a requirement is dangerous, meaning, fulfillment of the requirement is unrealizable or at least expensive. Use of universals in a domain assumption is fine because it simply states that the design will work only if the assumption holds. That is, such use of universals simply (and usefully) delimits the scope of the solution represented by the design.

VIII. CONCLUSION

In this paper, we treat NFRs as requirements over qualities, proposing a compositional language and a goal-oriented methodology to capture them. We start with quality mappings, and then discuss the domain and codomain of qualities: a collective subject can lead to ***universality***, a subjective one likely needs ***agreement***, and quality values versus desired quality regions will give rise to the ***gradability*** of NFRs [14]. We propose a goal-oriented methodology to refine early and informal NFRs to make them unambiguous, satisfiable and measurable. Our proposal serves to: (1) better understand what NFRs are; (2) better distinguish between NFRs and FRs; (3) write better NFR specifications.

Several issues remain open. One is the integration of contribution links with our formalism. As we have discussed, contribution links are indispensable because they capture the influences of functions or function constraints on quality goals. However, integration with our quality-based formalism has not yet been fully explored.

Another important issue is how to perform reasoning on our revisited goal models. In our framework, the satisfaction of a quality goal will depend on the membership between the measured quality value and the expected quality region. Moreover,

when a quality is refined to several sub-qualities, its co-domain will be a multi-dimensional space with each of its sub-quality as a dimension. As such, the reasoning over quality goal satisfaction in our framework will differ from classical goal model reasoning (e.g., [45]), and needs to be further investigated.

ACKNOWLEDGMENT

We are grateful to the anonymous reviewers for helpful suggestions that enabled us to much improve this paper in content and presentation. This research has been funded by the ERC advanced grant 267856 "Lucretius: Foundations for Software Evolution", unfolding during the period of April 2011 - March 2016. It is also financially supported by the National Natural Science Foundation of China (No. 61033006).

REFERENCES

[1] ISO/IEC 25010, "Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models," 2011.

[2] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," Software Engineering, IEEE Transactions on, vol. 18, no. 6, pp. 483–497, 1992.

[3] L. Chung, B. A. Nixon, and E. Yu, Non-Functional Requirements in Software Engineering, vol. 5. Kluwer Academic Pub, 2000.

[4] F.-L. Li, J. Horkoff, J. Mylopoulos, L. Liu, and A. Borgida, "Non-Functional Requirements Revisited," iStar, 2013, pp. 109–114.

[5] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari, "Ontology Library," WonderWeb Deliverable D18, 2003.

[6] G. Guizzardi, Ontological foundations for structural conceptual models. CTIT, Centre for Telematics and Information Technology, 2005.

[7] T. Menzies, B. Caglayan, H. Zhimin, K. Ekrem, K. Joe, P. Fayola, and T. Burak, "The PROMISE Repository of empirical software engineering data," Jun-2012. [Online].Available: http://promisedata.googlecode.com.

[8] "Feature structure," Wikipedia, the free encyclopedia. 16-Jan-2014.

[9] A. Dardenne, A. Van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," Science of computer programming, vol. 20, no. 1–2, pp. 3–50, 1993.

[10] E. S.-K. Yu, "MODELLING STRATEGIC RELATIONSHIPS FOR PROCESS REENGINEERING," University of Toronto, 1995.

[11] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling," RE, 2010, pp. 115–124.

[12] F. Dalpiaz, V. Silva Souza, and J. Mylopoulos, "The Many Faces of Operationalization in Goal-Oriented Requirements Engineering," APCCM, 2014.

[13] P. Gärdenfors, Conceptual spaces: The geometry of thought. MIT press, 2004.

[14] R. S. S. Guizzardi, F.-L. Li, A. Borgida, G. Guizzardi, J. Horkoff, and J. Mylopoulos, "An Ontological Interpretation of Non-Functional Requirements," FOIS, 2014.

[15] A. Albuquerque and G. Guizzardi, "An ontological foundation for conceptual modeling datatypes based on semantic reference spaces," RCIS, 2013, pp. 1–12.

[16] ISO/IEC 9126-1, "Software Engineering - Product Quality - Part 1: Quality Model," 2001.

[17] ISO/IEC 25030:2007, "Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Quality requirements."

[18] R. J. Brachman and H. J. Levesque, Knowledge representation and reasoning. Morgan Kaufmann, 2004.

[19] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation," RE, 2010, pp. 125–134.

[20] I. J. Jureta, S. Faulkner, and P.-Y. Schobbens, "Clear justification of modeling decisions for goal-oriented requirements engineering," Requirements Engineering, vol. 13, no. 2, pp. 87–115, 2008.

[21] ISO/IEC 9126-2, "Software engineering - Product quality - Part 2: External metrics," International Organization for Standardization, Geneva, Switzerland, 2003.

[22] P. Gärdenfors, Meanings as conceptual structures. Lund University, 1995.

[23] G. Elahi and E. Yu, "Modeling and analysis of security trade-offs–A goal oriented approach," Data & Knowledge Engineering, vol. 68, no. 7, pp. 579–598, 2009.

[24] B. Paech and D. Kerkow, "Non-functional requirements engineering-quality is essential," REFSQ, 2004.

[25] M. Glinz, "On non-functional requirements," RE, 2007, pp. 21–26.

[26] L. Chung and J. do Prado Leite, "On non-functional requirements in software engineering," in Conceptual Modeling: Foundations and Applications, 2009, pp. 363–379.

[27] I. C. S. S. E. S. Committee, I. Electronics Engineers, and I.-S. S. Board, IEEE recommended practice for software requirements specifications: approved 25 June 1998, vol. 830. IEEE, 1998.

[28] J. Robertson and S. Robertson, "Volere: Requirements specification template," Technical Report Edition 6.1, Atlantic Systems Guild, 2000.

[29] R. Berntsson Svensson, T. Olsson, and B. Regnell, "An investigation of how quality requirements are specified in industrial practice," Information and Software Technology, vol. 55, no. 7, pp. 1224–1236, 2013.

[30] S. Supakkul and L. Chung, "Integrating FRs and NFRs: A use case and goal driven approach," framework, vol. 6, p. 7, 2005.

[31] L. M. Cysneiros and J. C. S. do Prado Leite, "Using UML to reflect non-functional requirements," CASCON, 2001.

[32] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," Autonomous Agents and Multi-Agent Systems, vol. 8, no. 3, pp. 203–236, 2004.

[33] I. J. Jureta, J. Mylopoulos, and S. Faulkner, "Revisiting the core ontology and problem in requirements engineering," RE, 2008, pp. 71–80.

[34] I. Jureta, S. Faulkner, and P. Y. Schobbens, "A more expressive softgoal conceptualization for quality requirements analysis," ER, 2006, pp. 281–295.

[35] A. van Lamsweerde, "Goal-oriented requirements enginering: a roundtrip from research to practice [enginering read engineering]," RE, 2004, pp. 4–7.

[36] B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. MacLeod, and M. J. Merrit, Characteristics of software quality, vol. 1. North-Holland Publishing Company, 1978.

[37] R. B. Svensson, M. Host, and B. Regnell, "Managing quality requirements: A systematic review," SEAA, 2010, pp. 261–268.

[38] T. Gilb, Competitive engineering: a handbook for systems engineering, requirements engineering, and software engineering using Planguage. Butterworth-Heinemann, 2005.

[39] A. Cailliau and A. van Lamsweerde, "A probabilistic framework for goal-oriented risk analysis," RE, 2012, pp. 201–210.

[40] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, "Relax: Incorporating uncertainty into the specification of self-adaptive systems," RE, 2009, pp. 79–88.

[41] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Reasoning with goal models," ER, 2002, pp. 167–181, 2003.

[42] J. Yen and W. A. Tiao, "A systematic tradeoff analysis for conflicting imprecise requirements," RE, 1997, pp. 87–96.

[43] K. Pohl, "The three dimensions of requirements engineering," CAiSE, 1993, pp. 275–292.

[44] D. M. Berry and E. Kamsties, "The dangerous 'all' in specifications," IWSSD, 2000, pp. 191–193.

[45] J. Horkoff and E. Yu, "Comparison and evaluation of goal-oriented satisfaction analysis techniques," Requirements Engineering, pp. 1–24, 2012.