An Excursion to Ontology-Based Non-functional Requirements Specification

Krupa Patel[1] and Unnati Shah[2](✉)

[1] Bhagwan Mahavir University, Surat 395007, India
[2] Utica University, Utica, NY 13495, USA
unshah@utica.edu

Abstract. It is challenging to ask stakeholders to explicitly provide Non-Functional Requirements (NFRs) for the software due to the domain-specific and context-sensitive nature. A formal and unambiguous specification of NFRs is provided by creating an NFRs ontology based on domain knowledge and quality models. Ontology concepts make NFRs specification definite, complete, consistent, and convenient to share and reuse. In this paper, we intend to examine the approaches for specifying NFRs. The purpose of this study is to disseminate, analyze, and deconstruct the published research in the field, and select metrics for a comparative assessment.

Keywords: Non-Functional Requirements · Quality Model · Ontology

1 Introduction

Functional Requirements (FRs) and Non-Functional Requirements (NFRs) are essential to the success of any software [1]. It is easy to identify FRs from the Requirements Specification Document (RSD). While capturing NFRs in RSD is difficult due to hidden and vague nature. The NFRs from the RSD are often manually identified by users and requirements engineers utilizing their experience and skills [2]. However, as NFRs are constantly associated with certain domains and influenced by context, it is challenging to request users to submit NFRs directly. Additionally, it is challenging to adapt to the environment's changing requirements and to explain them uniformly and consistently. Therefore, the requirements engineer faces a hurdle when trying to specify NFRs without having a thorough understanding of them. Specification of NFRs and its common interpretation among different stakeholders are promising issues, among others. It is difficult to state NFRs explicitly due to lack of domain knowledge. The domain knowledge helps requirements analyst to ask relevant questions to specify all possible NFRs at early stage by means of ontology [3, 4].

Ontology is a formal description of the concept of sharing, stressing the link between real entities [3]. Because of interoperability and machine reasoning, ontologies have become more widely used in computing in recent years. An ontology is a taxonomy of domain terms. The term ontology implies the modeling of domain rules, which provide an extra level of machine understanding. A formal, explicit specification of a shared conceptualization is provided by the ontology's construction, which is based on domain knowledge and quality models [4]. It assists domain users in properly expressing their requirements, and it aids requirements analysts in accurately understanding and modeling the requirements. Additionally, RSD is definite, comprehensive, consistent, and easy to share and reuse because of the ontology concepts. There are several ontology-based NFRs specification methodologies in the literature [5–17]. The ontology can help developers have a shared understanding of NFRs and can serve as the foundation for NFRs standard.

2 Non-functional Requirements

"The hardest part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all of the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later". [18].

Eliciting, analyzing, and writing precise RSD are the most difficult parts of software engineering. There are two types of requirements viz. FRs and NFRs. FRs depend on end users, system actions and business needs. Contrary, NFRs depends on properties that the system should have and the way in which the customers want the product to act. (viz. performance, security, usability, etc.) [19]. The main issue is when all FRs are met but the overall software still fails to meet some requirements because NFRs are not managed. Consider an ATM, for instance, whose primary purpose is to allow users to withdraw cash from banks. What if a transaction takes the system two hours to complete? What if the account information is not safe? Today, it is impossible to imagine living without smartphones. The phones save all data, including personal, professional, banking, and other details. What if the cell phone data is not secure? The NFRs are therefore crucial to the success of any software. In the literature, there exist various definitions of NFRs as shown in Table 1.

Each NFR has different quality characteristics (QCs) and roles during the software development process. Consider security NFRs describe numerous system security features including authentication, authorization, and privacy, whereas usability NFRs discuss numerous facets of the system's usability and usability. There are 248 different types of NFRs in the literature, including QCs. Out of 248 types of NFRs, researchers deal with 106 NFRs, out of which 28 NFRs are defined with detailed QCs, 29 NFRs are defined without QCs, and 54 NFRs are introduced without definition. This implies that there is still no definition for 50.94% of NFRs in the literature.

Table 1. Definitions of NFRs

Source	Definition
[20]	*"The term is not defined. The standard distinguishes design requirements, implementation requirements, interface requirements, performance requirements, and physical requirements."*
[21]	*"The required overall attributes of the system, including portability, reliability, efficiency, human engineering, testability, understandability, and modifiability."*
[19]	*"These are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process and standards. They often apply to the system as a whole. They do not usually just apply to individual system features or services."*
[22]	*"Describe the non-behavioral aspects of a system, capturing the properties and constraints under which a system must operate."*
[1]	*"The standard defines the categories functionality, external interfaces, performance, attributes (portability, security, etc.), and design constraints."*
[23]	*"A requirement that specifies system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility, and reliability. A requirement that specifies physical constraints on a functional requirement."*
[24]	*"A property, or quality, that the product must have, such as an appearance, or a speed or accuracy property."*
[25]	*"The behavioral properties that the specified functions must have, such as performance, usability."*
[26]	*"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."*
[27]	*"There is not a formal definition or a complete list of nonfunctional requirements. They are global requirements on its development or operational cost, performance, reliability, maintainability, portability, robustness, etc. Informally stated, often contradictory, difficult to enforce during development and evaluate for the customer prior to delivery."*

3 Quality Models

A quality model is defined as *"the set of characteristics, and the relationships between them, that provides the basis for specifying quality requirements and evaluation"* by [29]. Models for evaluating software quality have been created, specifying the fundamental elements (also known as characteristics) and the sub factors within each of them. Each sub factor has a set of metrics for the actual evaluation. Quality model evolution is seen in [28]. This evolution has divided the models into two categories: 1. Basic Models, whose goal is to evaluate products holistically; and 2. Tailored Quality Models, which are focused on component evaluations. According to [1, 20] the degree to which a system, component, or process satisfies a user's wants or expectations is the same as the degree to which the system, component, or process satisfies specified requirements. The first published quality models for software date back to the late 1970s, when Boehm et al. [30] described quality characteristics and their decomposition.

These models have come in a number of modifications over time. The FURPS model, which divides quality into Functionality, Usability, Reliability, Performance, and Supportability, is one of the more well-known ones. The basic concept of these models, aside from this hierarchical decomposition, is that you deconstruct quality down to a level where you can measure and, hence, evaluate quality. The international standard ISO/IEC 9126 was based on these types of quality models. The standard proposes a few metrics for measuring them and establishes a standard decomposition into quality characteristics. However, these criteria do not account for all facets of quality. As a result, the standard does not guarantee full operational quality. The new standard ISO/IEC 25010, which replaces ISO/IEC 9126, modifies a few classifications but maintains the overall hierarchical breakdown.

These quality models are static and unreadable by machines. As a result, we discovered a number of methodologies in the literature that employ ontology to implement the quality models. The ontological quality models can be processed by a machine.

4 Ontology

The Greek words ontos (being) and logos (word) are the source of the word ontology. It disparages the study of being and the descriptions of how existence is arranged, identified, and categorized [31]. With the help of ontologies, individuals and application systems can communicate about a topic with a common understanding. There is an obvious link between the ontological modeling of a domain and the modeling that a requirements engineer would undertake during the requirements process since Requirements Engineering (RE) entails knowledge collecting and analysis. Nearly all RE formalisms are reduced to first-order logic, and they all require a specific conception. They thus share many characteristics with ontologies created using formal language. The following are some potential applications for building ontologies in RE: (i) Capture domain knowledge and offer a generally shared understanding of a subject. (ii). Allow interoperability and the reuse of domain knowledge by offering effective communication channels. (iii). Distinguish between operational and domain knowledge. (iv). Analyze subject-matter expertise. (v). Organize changeable and dynamic requirements.

In requirements engineering, authors [5] propose conceptual dependency of ontology, which illustrates the relationships between several ontologies, including requirements ontology, dependability ontology, and NFRs ontology. For capturing the fundamental stakeholder concerns during requirements engineering, including beliefs, desires, intentions, and evaluations, writers in [32] suggest a core ontology for requirements (dubbed CORE).

On the other hand, authors in [31] distinguish between three types of ontologies in RE: (i). Application domain ontology: It describes the business data needed to design the system, as well as the application domain knowledge. Understanding the domain aids in identifying dynamic and changing requirements. (ii). It captures the fundamental criteria, as well as their interdependencies and linkages. This ontology can be used to eliminate inadequate requirements definitions and reduce unclear requirements during requirements elicitation. An ontology for requirements can then be used for validation and verification. (iii). It describes the organization of requirements specification documents to help reduce the number of incomplete requirements specifications.

5 NFRs Specification

5.1 NFRs: A Hard or Soft Goal?

NFRs are considered as *quality* that take value in quality region either crisp (viz. 0–7 s) or vague (e.g., fast, slow). The crisp quality regions are defined as Quality Constraints (QCs). There is a need to identify which NFRs belongs in which quality hierarchy. The quality is not always stated explicitly; rather they are hidden in the requirements. Identifying those NFRs is an issue. In literature [27], NFRs are considered as a soft-goal. A soft goal means the requirements that have no clear-cut criteria of success. On the other hand, the hard goals (FRs) are considered as requirements having clear cut criteria of success. It is difficult to identify all situations that provide solutions to the soft-goal (vague requirements-can't decide the boundary) at an early stage of RE. In this case, we can identify the existence of the quality but cannot identify the scope of the quality. For example, *"design the system's main menu"* is a vague requirement and thus considered as a soft goal. After refining to NFRs goal, "*The menu buttons must have standard length and width."* yet the requirement is vague. The vague NFRs are continuously refined and operationalized, and hence such vagueness disappears [17].

On the other hand, some researchers claim that the NFRs/FRs could be hard-goal as well as soft goal, rather strictly NFRs as a soft goal and FRs as hard goal [32]. Furthermore, there exists no clear-cut boundary between NFRs and FRs and it is difficult to integrate NFRs with the FRs. In support to the claim, authors [14] provide precise definitions of soft-goal as a goal without a clear-cut definition of satisfaction, but an independent existence; Quality Goal as a quality requirement, which consists of a quality type, a desired quality region and a subject, with a different representation; Quality Constraints-precisely defines a desired quality region in the quality space of a quality type and is achieved or denied by tasks. They mention the incomplete ontology can be extended on demand. Furthermore, they present their ontology and axioms for satisfaction of quality goal.

The basic question is how to refine the NFRs to identify all possible solutions that satisfy them by means of measurement or remove the vagueness. The process of refinement conceptually deconstructs the NFR's qualities by means of identifying the qualities that are inherent in the quality associated with an NFR. For example, a *security* quality can be refined in its sub-qualities viz. *confidentiality*, *integrity* and *availability.* Another way to refine the NFRs is to identify whether the quality is a resultant quality, which can be conceptually reduced to the qualities of the parts of the bearer of the original quality. For example, consider a requirement, *"The user interface must have standard format"*. The requirement contains *"standard format"* as a quality that in terms parts of the bearer *"the interface"*. The interface is generally composed of buttons, fields, icons, etc., "*The menu buttons must have standard format*" illustrates a possible refinement of the requirement. The requirement can further refine because the button may decompose into "*size*", "*shape*", and "*color*". Considering "*size*", it may be further decomposed into "*height*" and "*width*". Hence, a further refinement is *"The menu buttons must have standard height and width"*.

Ultimately, we need to provide some values to make NFRs measurable that make NFRs in a quality constraint region known as operationalization NFRs. For example, the previous refined requirement can be rewritten with constraint as "*The menu buttons must have height 0.75 cm and width 1.75 cm*". It is not mandated that the qualities are constrained to have specific quality values, rather it may concern a region. For example, requirement "*The search functionality must be efficient*", operationalized by "*The search results shall be returned within 30 s after the user enters search criteria*". For instance, take the requirement *"Learning to operate the login functionality must be efficient"*. This NFR may be operationalized by "*The user should learn how to operate the login functionality within 3 min*". Thus "*efficient*" for learning the login functionality and for returning search results (previous example) may map to different regions in the time quality dimension.

5.2 NFRs Extraction: Interview-Based Approach to Ontology-Based Approach

At first stage requirements analyst extract NFRs by means of interview. At this stage NFRs are unclear and unstructured. Furthermore, requirements analyst may not have complete knowledge regarding the domain specific NFRs. Therefore, the requirements analyst must be supported by means of knowledge that will help them to identify pertinent NFRs and to ask right questions while eliciting NFRs at the time of interview. In literature, there exists a handful of proposals that give conceptual representation of NFRs. Much of the awareness about NFRs comes from the original work present in [27]. The NFR Framework is a goal-oriented approach, used to represent NFRs graphically by means of soft-goal interdependency graph. It provides methods of refinement, to further refine a goal into one or more goals.

To reduce the problem of integration, in [2], NFRs are integrated by means of extending UML models. However, this approach models certain NFRs due to limited knowledge regarding quality attributes. Instead, UML models, natural language processing helps to extract NFRs from requirements documents. Lee et al. [11] apply natural language processing to provide a common understanding of security requirements and to facilitate analysis at various decision points by making the required information readily available with appropriate context and format. However, they focus only on security requirements. Furthermore, natural language processing cannot help to provide additional information regarding application domain. NFRs are difficult to describe completely and precisely due to vague, conceptual, and subjective nature [11].

To identify NFRs there is a need to provide the domain knowledge support at the time of interview. Different NFRs has different meaning so ontology provides common understanding of the terms [10]. The authors [6] developed two ontology, one is to extract FRs using domain knowledge named functional domain ontology and another is to extract NFRs using standard quality model-ISO/IEC 9126 named quality ontology. To support the ontology the authors developed an interactive tool-ElicitO. They gather knowledge from textbooks and experts in the domain to build the functional ontology [6]. Furthermore, they present a scenario where the user and analyst seat together and use the tool to extract requirements, meanwhile using ontology appropriate NFRs are also considered that provide standardization compared to traditional approaches. To build the ontology authors follow the process presented in [7].

The generalized approach can be appropriate for any domain, but the quality level and the order of importance may vary from one domain to another. The aim of developed ontology is to provide the definition of the general concepts relevant to NFRs without reference to any domain. These general concepts can then act as a common foundation for describing NFRs attributes, metrics, etc. Furthermore, it provides a conceptual model for NFRs that contains rules which define the semantics of the defined concepts.

To provide the more generic solution, the concept of ontology used in [5, 8–10] in order to develop software rapidly. That is, common requirements in a specific domain should be treated as a special kind of reusable resources. In [8] authors explain the need of machine understandable Quality of Service (QoS) vocabulary and importance of sharing it for both static and dynamic systems architectures. The authors developed a tool named QoSOnt that provides an extension to OWL-S (non-Qos service provider). They focus on developing ontology for quality parameters viz. dependability, reliability and performance supporting metric representation. They build the complete ontology by means of interconnecting "base ontology" - separate ontology for all concept attributes of quality. The reason behind the separation is to refer the particular metrics of that quality attribute. In addition, they create a tool named SQRM to evaluate the validity of generated ontology.

One NFR can help or hinder the achievement of other NFRs at certain functionality. To identify such interaction manually is difficult. To make the process automate authors [3] focus not only extraction of NFRs, they provide relation of NFRs by means of identifying NFRs from different view perspective. They present three NFRs ontology views: 1. Intra-Model Dependency - NFRs relation with the other entities of the software system being developed. 2. Inter Model Dependency - contains the classes and properties intended to structure NFRs in terms of interdependent entities. 3. Measurable NFRs - measurement process and contains the concepts used to produce measures to measurable NFRs.

However, NFRs are not independent goals, rather they are related to other parts of software. If the requirements are classified as the NFRs it is necessary, that it belongs to at least one FR as it represents a set of attributes that bear on the existence of a set of functions and their specified properties according to the ISO9126 definition. For example, *"The interaction between the user and the software system while reading email messages must be secured"*. Here, reading email message is FR, while providing a secure environment while reading a message is NFRs.

5.3 Comparative Analysis and Issues

Table 2 presents a comparative study of existing approaches for specifying NFRs, considering parameters including elicitation, categorization, input, output, quality model, and dataset. Majority tools [5–17] uses portege as a ontology development tool. We observe that there exists no tool that provides validation of extracted NFRs except efforts given by [11]. Generally, the input is natural language. Most of the tools use input as a natural language except [3] uses XML representation of work flow as an input. A Service-Centric System (SCA) may have a static architecture decided at design-time.

While Service-Oriented Architectures (SOA) also open up the possibility of a dynamic architecture where certain components are only discovered and bound at runtime. Both approaches ideally require a service marketplace, in which functionally equivalent services compete for the same custom. In such a situation non-functional characteristic, i.e. QoS, become more important in distinguishing between the competing services. In either case, it is also desirable to have a machine understandable QoS vocabulary. In the static case this would prove useful in implementing design-time tools, in the dynamic case it is necessary to perform service selection at runtime based upon service QoS.

Table 2. Comparative Analysis of Approaches to Specify NFRs[1]

Citation	Elicit	Classify	Input	Output	Quality Model	Dataset
[5]	NFRs	Y	NL	NL	Generic	AutomotiveIndustry
[6]	FRs+NFRs	Y	NL	NL	ISO/IEC 9126	UniversityHelpdesk
[7]	FRs+NFRs	N	NL	Graph	Generic	AutomotiveIndustry
[9]	FRs+NFRs	Y	NL	NL	ISO/IEC 9126	UniversityHelpdesk
[10]	NFRs	N	XML	Graph	NFRs	Web Services
[3]	NFRs	Y	NL	Ontology	Generic	Email Application
[11]	Security	Y	NL	Ontology	Generic	Website
[13]	FRs+NFRs	Y	NL	NL	ISO9126	Concordia RE Corpus
[14]	FRs+NFRs	N	NL	Ontology	Generic	Urban Transportation Management
[15]	NFRs	Y	NL	Ontology	ISO/IEC25010	Promise
[16]	FRs+NFRs	Y	NL	Modeling language	ISO/IEC 9126	Promise
[17]	NFRs	Y	NL	Ontology	ISO/IEC25010	Promise

Ontologies are regarded as a valuable component of RE, although there are several challenges when creating ontology-based software development projects, including maintenance, merging, mapping, and versioning of ontologies. It is difficult to determine the “correct” level of expressiveness for the ontology language. Furthermore, there are numerous distinct ontology languages that are used for various reasons, and it is not immediately obvious how to map between ontologies that are expressed using various languages. In Table 3, we identify issues specifying NFRs using ontology.

[1] Y – Yes; N – No; NL – Natural Language; XML – Extensible Markup Language.

Table 3. Issues in Specifying NFRs using Ontology

Issue	Description
NFRs Conflict	It can happen that one NFR has a favorable or unfavorable impact on subsequent NFRs. Finding the conflict between the specified NFRs is necessary to provide the conceptualization
NFRs Traceability	One NFR may affect many functional requirements. While changing requirements affect both the functional and NFRs. There is a need to trace the relationship exist among them
NFRs Integration	How NFRs integrate with functional requirements?
NFRs Ambiguity	The representation of ontology gives a common interpretation among requirements engineers. The problem of ambiguity is still an issue from the customer's perspective
NFRs Modeling	How to represent NFRs effectively at requirements design stage?

6 Conclusions

In this paper, the concepts and quality models of the NFRs are thoroughly examined. The research also discusses a comparative examination of alternative ontology-based approaches to specify NFRs. The ontology has become more prominent in RE study areas, with a focus on defining NFRs. However, there are issues with ontology that are related to their application, such as merging, mapping, integrating, etc. Future research will focus on looking at potential solutions to the issues related to specifying NFRs such as conflict, traceability, ambiguity, and modeling.

References

1. Software Engineering Standards Committee: IEEE Recommended Practice for Software Requirements Specifications (1998)
2. Mijanur Rahman, M., Ripon, S.: Elicitation and modeling non-functional requirements – a POS case study. Int. J. Future Comput. Commun. **2**(5), 485–489 (2013)
3. Kassab, M., Ormandjieva, O., Daneva, M.: An ontology based approach to non-functional requirements conceptualization. In: 2009 Fourth International Conference on Software Engineering Advances, pp. 299–308 (2009)
4. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. **5**(2), 199–220 (1993)
5. Dobson, G., Hall, S., Kotonya, G.: A domain-independent ontology for non-functional requirements. In: IEEE International Conference on E-business Engineering, pp. 563–566 (2007)
6. Al Balushi, T.H., Sampaio, P.R.F., Dabhi, D., Loucopoulos, P.: ElicitO: a quality ontology-guided NFR elicitation tool. In: Sawyer, P., Paech, B., Heymans, P. (eds.) REFSQ 2007. LNCS, vol. 4542, pp. 306–319. Springer, Heidelberg (2007).https://doi.org/10.1007/978-3-540-73031-6_23
7. Falbo, Rd.A., de Menezes, C.S., da Rocha, A.R.C.: A systematic approach for building ontologies. In: Coelho, H. (ed.) IBERAMIA 1998. LNCS (LNAI), vol. 1484, pp. 349–360. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49795-1_31

8. Dobson, G., Lock, R., Sommerville, I.: Quality of service requirements specification using an ontology. In: SOCCER Workshop, Requirements Engineering, vol. 5 (2005)
9. Al Balushi, T.H., Sampaio, P.R.F., Dabhi, D., Loucopoulos, P.: Performing requirements elicitation activities supported by quality ontologies. In: 18th International Conference on Software Engineering and Knowledge Engineering, SEKE 2006, no. August 2015, pp. 343–348 (2006)
10. Hughes, C., Hillman, J.: QoS explorer: a tool for exploring QoS in composed services. In: International Conference on Web Services, ICWS 2006, pp. 797–806 (2006)
11. Lee, S.-W., Muthurajan, D., Gandhi, R.A., Yavagal, D., Ahn, G.-J.: Building decision support problem domain ontology from natural language requirements for software assurance. Int. J. Softw. Eng. Knowl. Eng. **16**(6), 851–884 (2006)
12. Li, F.-L., Horkoff, J., Mylopoulos, J., Liu, L., Borgida, A.: Non-functional requirements revisited. In: Proceedings of the 6th International i* Workshop, vol. 978, no. iStar, pp. 109–114 (2013)
13. Rashwan, A., Ormandjieva, O., Witte, R.: Ontology-based classification of non-functional requirements in software specifications: a new corpus and SVM-based classifier. In: COMPSAC 2013 Proceedings of the 2013 IEEE 37th Annual Computer Software and Applications Conference, no. ii, pp. 381–386 (2013)
14. Jingbai, T., Keqing, H., Chong, W., Wei, L.: A context awareness non-functional requirements metamodel based on domain ontology. In: Proceedings - 1st IEEE International Workshop on Semantic Computing and Systems, WSCS 2008, pp. 1–7 (2008)
15. Li, F.-L., et al.: Non-functional requirements as qualities, with a spice of ontology. In: Proceedings of the 22nd International Requirements Engineering Conference, pp. 293–302 (2014)
16. Li, F.-L., Horkoff, J., Borgida, A., Guizzardi, G., Liu, L., Mylopoulos, J.: From stakeholder requirements to formal specifications through refinement. In: Fricker, S.A., Schneider, K. (eds.) REFSQ 2015. LNCS, vol. 9013, pp. 164–180. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16101-3_11
17. Guizzardi, R., Li, F., Borgida, A., Guizzardi, G.: An ontological interpretation of non-functional requirements. In: Formal Ontology in Information Systems, pp. 344–357. IOS Press (2014)
18. Brooks, F.P., Bullet, N.S.: Essence and accidents of software engineering. Computer **20**, 10–19 (1987)
19. Sommerville, I.: Software Engineering, 5th edn (1995)
20. IEEE Standard: IEEE Standard Glossary of Software Engineering Terminology (1990)
21. Davis, A.M.: Software Requirements: Objects, Functions, and States. Prentice-Hall Inc., Upper Saddle River (1993)
22. Anton, A.I.: Goal Identification and Refinement in the Specification of Software-based Information Systems. Georgia Institute of Technology, Atlanta, GA, USA (1997)
23. G. Jacobson, I., Booch, G., Rumbaugh, J., Rumbaugh, J., & Booch, I. Jacobson, G. Booch, and J. Rumbaugh. (1999)
24. Robertson, S., Robertson, J.: Mastering the Requirements Process (1999)
25. Ncube, C.: A requirements engineering method for COTS-based systems development (2000)
26. Karl, W.: Software requirements (2003)
27. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Springer, New York (2012)
28. Miguel, J.P., Mauricio, D., Rodríguez, G.: A review of software quality models for the evaluation of software products. Int. J. Softw. Eng. Appl. **5**(6), 31–53 (2014)
29. IEC 9126-1: Software Engineering-Product Quality-Part 1: Quality Model (2001)

30. Boehm, B.W., Brown, J.R., Lipow, M.: Quantitative evaluation of software quality. In: Proceedings of the 2nd International Conference on Software Engineering, pp. 592–605 (1976)
31. Castaneda, V., Ballejos, L., Caliusco, L., Galli, R.: The use of ontologies in requirements. Glob. J. Res. Eng. **10**(6), 2–8 (2010)
32. Jureta, I.J., Mylopoulos, J., Faulkner, S.: Revisiting the core ontology and problem in requirements engineering. In: Proceedings of the 16th IEEE International Requirements Engineering Conference, RE 2008, no. November 2015, pp. 71–80 (2008)