Towards a Scope Management of Non-Functional Requirements in Requirements Engineering

M. Kassab[1], O. Ormandjieva[2] , M. Daneva[3]
[1,2]*{moh_kass , ormandj }@ cse.concordia.ca*
[3]*m.daneva @ utwente.nl*

Abstract. *Getting business stakeholders' goals formulated clearly and project scope defined realistically increases the chance of success for any application development process. As a consequence, stakeholders at early project stages acquire as much as possible knowledge about the requirements, their risk estimates and their prioritization. Current industrial practice suggests that in most software projects this scope assessment is performed on the user's functional requirements (FRs), while the non-functional requirements (NFRs) remain, by and large, ignored. However, the increasing software complexity and competition in the software industry has highlighted the need to consider NFRs as an integral part of software modeling and development. This paper contributes towards harmonizing the need to build the functional behavior of a system with the need to model the associated NFRs while maintaining a scope management for NFRs. The paper presents a systematic and precisely defined model towards an early integration of NFRs within the requirements engineering (RE). Early experiences with the model indicate its ability to facilitate the process of acquiring the knowledge on the priority and risk of NFRs.*

1. Introduction

In order to meet commitments in software projects, a realistic assessment must be made of project scope. Such an assessment relies on (i) availability of knowledge on the user-defined project requirements, (ii) their effort estimates, (iii) their priorities, (iv) as well as their risk. This knowledge enables analysts, managers and software engineers to identify the most significant requirements from the list of requirements initially defined by the user. For instance, a requirement deemed critical, but which takes a great deal of implementation effort and poses a high risk, may be a good candidate for immediate resourcing.

In most software projects and industrial practices, this scope assessment is performed on the user's functional requirements (FRs), while the non-functional requirements (NFRs) remain, by and large, ignored. However, the increasing software complexity and competition in the software industry has highlighted the need to consider NFRs as an integral part of software modeling and development. Empirical reports [1, 2, 3] consistently indicate that not considering NFRs in the early phases of the software development process leads to project failures, or at least to considerable delays, and, consequently, to significant increases in the final cost.

The goal of this research contributes to the above discussion as it aims at integrating NFRs earlier in the software development. This paper presents a systematic and precisely defined model towards the integration of NFRs within the Requirements Engineering (RE) process. The model is designed to facilitate the acquisition of knowledge on NFRs and to enhance their scope management through proper assignment for their priority and risks.

Our proposed model is depicted in Figure 1. The model represents a process composed of three phases: (i) Requirements Elicitation, (ii) Analysis and Crosscutting Realization and (iii) Composing Requirements. We use the term phase to describe a group of one or more activities within the model. The phase is a mean to categorize activities based on the general target they tend to achieve.

In this rest of this paper, we present related work in section 2, and then we describe each phase in a separate section (from 3 to 5). We conclude our work in section 6.

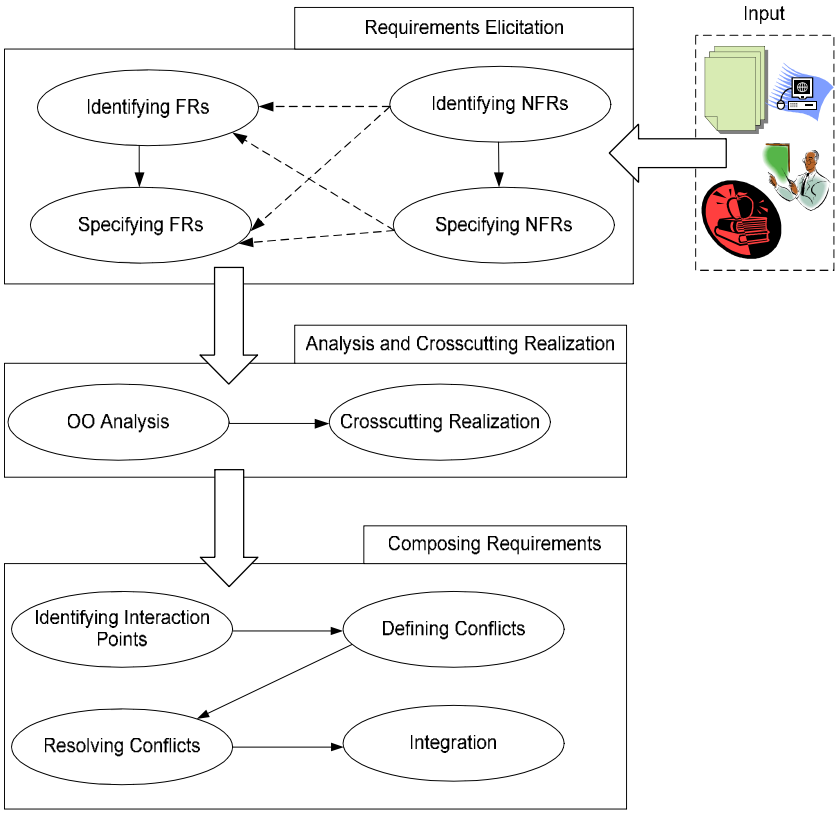

Figure 1: Proposed Model to integrate NFRs early in the RE

2. Related Work

The NFR framework [4] has been the first to propose the concept of softgoal to represent NFRs in the RE context. The NFR framework suggests that architects: (i) identify those NFRs that are vital to the system's success as *critical*; and (ii) identify NFRs that deal with a significant portion of the organization's workload as *dominant*. Missing from this approach, however, is (i) the impact that the stakeholders can have on the requirements elicitation process and (ii) the objective reasoning in the decision making process to select from different candidate operationalizations to satisfice NFRs. In [5], Cysneiros and Leite researched the process to elicit NFRs, analyzed their interdependencies, and traced them to functional conceptual models. They brought extensions of UML conceptual models which include a way to express NFRs. The key claim the authors made was that augmenting conceptual models with representations of NFRs can improve the quality of the resulting conceptual models themselves.

In [6], Robinson et al put in perspective the metrics-based Root Requirements Analysis (RRA) technique to confront the requirements interaction problem, which is how to discover, track and resolve conflicting interactions among NFRs. The authors define "a root requirement" as an abstract requirement that implies significant interactions—typically, conflicts. The RRA process is supposed to support (1) analysis of requirements interactions, and (2) ordering requirements by their degree of conflict. Having this information, the authors demonstrate how an architect can iteratively resolve conflicts in a large requirements document by applying four

steps: (1) structure the requirements, (2) identify root requirements, (3) identify central interactions, and (4) iteratively resolve conflicts.

Last, related work on NFRs prioritization includes [7,8,9,10,11,12]. These sources suggest that the NFRs' priority is to be assigned subjectively by stakeholders to reflect business criticality, importance to the customers or users, urgency, importance for the product architecture, or fit to release theme.

3. Requirements Elicitation Phase

The first phase in the proposed model is requirements elicitation. This phase aims to discover the requirements for the system. It is composed of four activities: identifying FRs, specifying FRs, identifying NFRs and specifying NFRs.

Identifying FRs: Functional requirements capture the intended usage of the system. This usage may be expressed as services, tasks or functions which the system is required to perform. A context diagram could be an excellent starting point for capturing the system's boundaries, users and FRs. Identifying FRs is an activity that involves discussions with stakeholders, reviewing proposals, building prototypes and arranging requirements elicitation meetings.

Specifying FRs: In this activity, we further refine each usage of the system into a detailed functional behavior described as a use-case with textual description. Thus, each FR is mapped to one or more use-cases. The outcome is the completion of a use-case description for each use-case (Table 1). Table 1 is similar to the fully dressed format suggested by Larman in [13].

Table 1. Template to Specify Use-Cases

Use Case No.	Unique to the use-case.
Name	The name of the use-case.
Priority	Importance of the use-case.
Actors	Primary and secondary actors.
Precondition	Textual description of the condition that must be satisfied before the use-case is executed.
Main scenario	A single and complete sequence of steps describing an interaction between a user and a system.
Alternative scenario	Extensions or alternate courses of main scenario.
Postcondition	Textual description of the condition that must be satisfied after the use case is executed.
Related Use Cases	Use-cases related to the current use- case.

Identifying NFRs: NFRs that are relevant to the problem domain are captured in parallel to the identification of FRs. While elicitation of NFRs can be accomplished by a number of existing techniques, the most recognized technique is to use NFRs catalog [4] where each entry in the catalog is cross listed against the decision of whether it is applicable for the system or not.

Specifying NFRs: Since NFRs often invite many different interpretations from different people, they need to be clarified as much as possible through refinements in discussions with the stakeholders. The stakeholders represent NFRs explicitly as softgoals to be satisfied. We propose the adoption of a matrix (Table 2) that relates the identified and specified NFRs to the FRs and use-cases they affect. In an invoicing system example, where we have a "Search for Product" functionality that must be provided by the system to the customer in a secure way, security is identified as an NFR that is placed as a constraint on the FR (and eventually on the use-case) "Search". Specifying the NFR "security" further, will decompose it down into three softgoals: confidentiality, availability and integrity. Depending on the requirements, we could be only concerned with availability as a constraint to be implied on "Search" functionality and thus the corresponding cell in the matrix is to be checked. In the case where an NFR would affect the system as a whole (e.g. portability), all entries in the corresponding column must be checked.

Table 2: Matrix to relate NFRs to FRs and use-cases

		NFR_1	NFR_2		…	NFR_n
			NFR_{2-1}	NFR_{2-2}		
FR_1	Use-Case1	√				
FR_2	Use-$Case_2$	√	√			
	Use-$Case_3$			√		
…						
FR_i	Use-$Case_m$					

While the notion of the softgoal to represent NFRs is dominant within the RE community, some NFRs are stated with quantitative terms which implies that the satisfaction for these NFRs has to be absolute and not relative. For example, Stakeholder may state a performance requirement as "The system shall respond within 3 seconds". This situation calls for extending the taxonomy of the NFR framework so that it can identify those NFRs that need to be stated in terms of crisp indicators and the acceptable values. We discuss our proposed extension in the following subsection.

3.1 Hardgoal Notation

Approaching the specification of NFRs through the NFR framework makes sense for stakeholder' requirements when they describe the system qualities they want built in laymen's terms, qualitatively citing specifications with accompanying verbal descriptions of how the functionality should be used [7]. However, for the satisfaction method to be performed on more concrete basis, stakeholders may agree to identify the NFRs with crisp indicators with defined acceptable values for those indicators to be satisfied; for instance, a performance requirement may be specified as follows: "The system shall be *of a high performance as the response time should be within 3 seconds."* This NFR describes a verifiable criterion (through the acceptable value of 3 seconds) for testing the system's response time quality. The NFR framework treats all NFRs as softgoals, but, as NFRs tend to be identified with crisp indicators, they are no longer soft and thus they should be modeled as "hardgoals". This is an omission from the NFR framework. Furthermore, there is a tight connection between the uses of the NFR's crisp indicators on the one hand, and the

quality of the process of acquiring knowledge on effort estimates, priority and risk on the other [14]. Having NFRs concretely defined in terms of the indicators leads to a more realistic assessment. Having performed the assessment, project decision-makers may then call for a revision of the crisp indicators to adjust the acceptable values. For example, if response time is a high priority for the system, then the acceptable value for the response time may need to be adjusted.

In response to this need, we propose an extension of the NFR framework, and its softgoal notation. Hardgoal is depicted with a star notation as shown in Figure 2. The graph in Figure 2 shows a performance softgoal with the new condition on response time and the imposed architecture constraint. NFR hardgoals are named using *Type[Topic1, Topic2,…]{Condition1, Condition2,…}* nomenclature. In our extension of the NFR framework's earlier notation, *Condition* indicates a relation to the acceptable values to verify the satisfaction level on NFR achievement.

One of the usages of hardgoals is to cope with the need to making it explicit the availability of knowledge on the user-defined requirements. For NFRs to be verifiablewhen the software is deployed, they have to be clearly defined in terms of crisp indicators. We term these clearly defined NFRs hardgoals. Defining the NFR as a softgoal could serve as an initial step towards our understanding for the required NFRs for the system, but eventually the softgoal are to be defined further and thus be presented as a hardgoal.

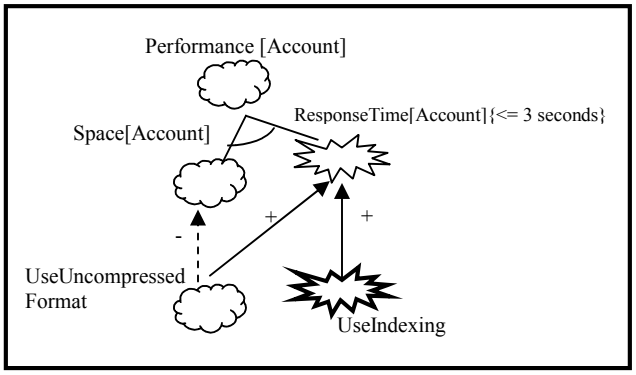

Figure 2: Employing the hardgoal notation in the NFR framework.

3.2 Priority

For design staff to be able to focus their effort on the most important NFRs, stakeholders should provide at the beginning of the RE process, their input on those NFR which are critical and, thus, are needed to proceed with its implementation first. The prioritization decisions can also be biased for some social, political or technical reasons. The NFR framework [4] suggests that architects: (i) identify those NFRs that are vital to the system's success as *critical*; and (ii) identify NFRs that deal with a significant portion of the organization's workload as *dominant*. In the framework, NFRs with high priority are identified by an exclamation point (!). The NFR framework deals with prioritization on qualitative basis. It is very important to notice that this initial prioritization should be done without too much influence from the developers involved in the downstream project activities; otherwise, the level of difficulty in implementing the goals will influence stakeholder priorities. The developers' input is accounted for in the execution-priority.

4. Analysis and Crosscutting Realization

Software requirements analysis is a critical phase of the software development process, as errors at this stage inevitably lead to later problems in the system design and implementation. In our model, the analysis phase is composed of two activities: OO Analysis and Crosscutting realization.

OO Analysis: The objective of the OO analysis activity is to understand the textual descriptions (requirements) that have been inducted in previous activities and to abstract the software under development into an OO analysis model. Analysis modeling is the formal or semi-formal presentation of the specification, through which the knowledge and information included in the textual description of the requirements are transmitted to the elements of the OO analysis model. The appropriate elements for OO analysis modeling are: use-case model diagrams, System Sequence Diagrams (SSDs), domain model diagrams, activity diagrams and state charts. In this discussion, we choose to focus on the first three diagram types to present the static and dynamic visions of the system. A domain model represents the static view and it illustrates meaningful (to other modelers) conceptual classes in a problem domain; it is the most important artifact to create during the OO analysis [13].

On other hand, at this activity, we model each sequence of events that are mapped from successful scenarios through an SSD. An SSD treats a system as a black box, placing emphasis on events that cross the system boundary from actors to the system and vice versa. The set of all required system operations within a SSD is determined by identifying the system events.

Crosscutting Realization: To identify the crosscutting nature of certain use-cases we need to take into consideration the information contained in Related Use Cases row in Table 1. If a use-case is included as a related use-case in several use-cases, then it is crosscutting.

On the other hand, the identified NFRs are classified as crosscuttings as they are considered as global properties of the system and they always crosscut at different spots of it. In this phase, crosscutting requirements are not modeled. They are only identified. These requirements will be modeled at the integration activity during composing requirements phase.

5. Composing Requirements

The goal of the phase of composing requirements is to integrate identified crosscuttings (both functional and nonfunctional) with the use-case model and the domain model. This is achieved in a series of four activities: (1) identifying the interaction points at which crosscutting requirements affect the system, (2) identifying possible conflicts among requirements at each interaction point, (3) resolving conflicts, and (4) integrating requirements.

Identifying Interaction Points: Based on requirements crosscuttings, we can identify interaction points in the system where crosscuttings will manifest themselves. We start by defining the set of requirements R = {FR}∪{NFR}, and the set of crosscuttings C = {Crosscutting requirements (CCRs)} $\subseteq$ R. We also define a function *A* which maps R to sets of CCRs as *A*: R$\rightarrow$P(C), where P states for "Powerset". The function *A* tracks those requirements that transverse several other requirements captured by this level of the development cycle.

Let $r \in R$, $c \subseteq C$. We define A as: $A(r) = \emptyset$, if there are no crosscutting requirements at r, and $A(r) = c$, otherwise. The set of Interaction Points I is defined as: $I = R - \{r \mid A(r) = \emptyset\}$.

Defining Conflicts: Hardly any requirements manifest in isolation, and normally the provision of one NFR may affect the level of provision of another. We refer to this mutual dependency as non-orthogonality. We propose a function M to map each pair of of the identified cross cutting requirements (CCRs) to values "+", "-" or "". M: $\{(CCR_i, CCR_j)\} \rightarrow$ {"+", "-", ""}. The rules for assigning the signs to the pairs of NFRs are as follows:

1. The value "-" is assigned to a pair of CCRs originating from the set of CCRs that contribute negatively at the same functionality. This means that one NFR in the pair has a negative (damage) effect on the other at the same functionality. The assignment is based on the experts' judgment of the developers. This is a case of a conflict between CCRs.

2. The value "+" is assigned to a pair of CCRs originating from the set of CCRs that contribute positively if they meet at the same functionality. This means that one CCR in the pair has a positive (constructive) effect on the other. The assignment is based on the experts' judgment of the developers.

3. The value "" is assigned to a pair of CCRs among the ones in the set of CCRs that do not interact. This assignment is based on the experts' judgment of the developers.

Resolving Conflicts: For each interaction point $P_i \in I$ we analyze the set $c = A(P_i)$, and study the contribution among its elements. We are essentially interested in those elements (requirements) that have a mutual negative interaction. We manage conflict resolution by assigning priorities of execution of the crosscuttings by mapping $A(P_i)$ to a sequence C_{seq}, where $P_i \in I$. An element in the sequence is either a crosscutting or a set of crosscuttings. The set notation within C_{seq} indicates that the elements within "{ }" are free to execute in any order relative to this position in the sequence, as there is no negative contribution identified. The process of mapping is guided by the expert's opinion. In the invoicing system example, we could have A (Place Order) = {Availability Confidentiality, Response Time, Process Payment}. But since Confidentiality and Availability have a mutual negative interaction between each other, we have to map the set to a sequence with an assigned order of execution. The result could be similar to [Confidentiality, Availability, {Response Time, Process Payment}]. The set element indicated that Response Time and Place Payment are free to execute in any order after Confidentiality and Availability.

On other hand, we observe that the approach of attributing a weight of significance to NFRs in order to identify dominance is not always applicable. In complex systems (such as concurrent systems) two or more NFRs may affect the same functionality with changing priorities with respect to the execution of the behavior of some component (e.g. method body), so assigning a hard-coded prioritization will not follow the correct semantics. For example, we may have a case with synchronization "sync" and scheduling "sched" whereby <sync, sched> method body<sched, sync> [15]. If authentication is introduced in the system, then priorities also change: <authentication, sync, sched>method-body<sched, sync, authentication>.

In addition, this approach of conflict resolution requires a major involvement of stakeholders. This makes it costly and dependent on stakeholder's availability. Moreover, in contrast to developers, business stakeholders are not interested in such system concerns and they may not have the necessary expertise to feel comfortable to get involved in these matters. They

would merely want their requirements implemented. These issues will be investigated further in the future research work.

Integration: In the integration activity, we compose and model all requirements based on the collected information from previous activities. Currently, there are many proposals in the literature on integrating the NFRs with UML [16, 17, and 18]. In our model, we extend the standard UML use-case diagram with a new stereotype <<CCR>> to abstract the crosscuttings integrated into the model, and use the <<include>> relation stereotype to indicate which use-cases are crosscut by the crosscuttings (see Figure 3). We extend domain model to include all NFRs that have been elicited earlier.

The knowledge required for creating the extended use-case model is extracted from output of function *A* against each defined requirement. In Figure 3,
use-case1, use-case2 $\in$ I and
Cross_Cutting_Requirement_1 $\in$ (A(use-case1) $\cap$ A(use-case2)) .

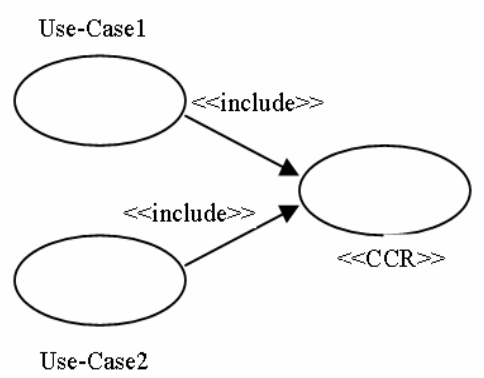

Figure 3: Integrated Use-Case model

5.1 Risk

Software industry has recognized risk management as a best practice for reducing the surprise factors in software projects. Risk management is about how to act early before a concern evolves into a major crisis.

In this section, we propose a quantitative risk assessment based on NFRs mutual dependency to identify potential risk in dealing with conflicting NFRs. We state that the negative interaction at run-time poses a risk to be considered. This is the case when two or more NFRs affect the same functionality and they interact in a negative way among each other during execution time. This happens mainly because the effort required for the integration process would highly depend on the level of interdependency between the NFRs, and, more specifically, on the defined conflicts among them. To objectively assess NFR conflicts, we propose to use the local conflict measure [19]. It reports on the level of conflict LLC (Local Level of Conflict) for each piece of functionality based on the list of NFRs that interact at this functionality:

$$LLC(f) = \left| \{(NFR_k, NFR_l) \bullet NFR_k, NFR_l \in NFRs_{at\_f} \wedge \quad M(NFR_k, NFR_l) = \text{"–"}\} \right| / n$$

In this formula, *n* is the cardinality of the set of all pairs of NFRs at functionality f (the order is ignored to avoid duplications). We can relate complexity of an arbitrary functionality to other functionalities complexities in the system using the following formula:

$$\text{Complexity}(f) = \left| \{(NFR_k, NFR_l) \bullet NFR_k, NFR_l \in NFRs_{\text{at } f} \wedge M(NFR_k, NFR_l) = \text{"}-\text{"}\} \right| / \sum_{j=1}^{n} \left| \{(NFR_k, NFR_l) \bullet NFR_q, NFR_r \in \{NFR_j\} \wedge M(NFR_q, NFR_r) = \text{"}-\text{"}\} \right|$$

The proposed measurements help in obtaining quantitative data that are supposed to direct the effort towards better design strategies and decisions. For example, high complexity values identify those pieces of functionality that pose more risk to the project; these pieces can be closely reexamined by architects to see which combinations of possible architectural options provide the best match; consequently, project managers may decide that more human resources, time or money needs to be dedicated to developing those pieces of functionality.

The collected quantitative data on NFRs provides the bases on which the stakeholders can plan for actions on how to minimize the likelihood or impact of these potential problems. Like the risk management due to FRs, NFR risk assessment makes it possible to focus on controlling the most serious risks first, thereby achieving better scope management of the requirements.

6. Conclusion

Existing approaches to handle NFRs come short when characterizing and quantifying hardgoal NFRs. These approaches adequately address primarily the softgoal NFR. They also lack quantitative support for objective analysis and decision making. We propose a solution to these issues and elaborate an extension to the NFR framework to allow modeling and analysis of hardgoal NFRs. In this paper, we also discussed a sequence of systematic activities towards an early consideration of identifying, specifying and separating FRs and NFRs. We provided a discussion on NFRs prioritization and risk assessment during the RE. For the purpose of achieving a concrete realization of the model, we intend to formalize the specification of the model's semantics.. Our future work will introduce a new formal language with an ontology designed to represent the primitive concepts that will be adequate for describing the model.

References:

1. Finkelstein, A., and Dowell, J., *A Comedy of Errors: The London Ambulance Service Case Study*, Proc. Eighth International Workshop Software Spec and Design, (1996), pp. 2-5.
2. Breitman, K. K, Leite J. C. S. P. and Finkelstein A, *The World's Stage: A Survey on Requirements Engineering Using a Real-Life Case Study*. *Journal of the Brazilian Computer Society* No 1 Vol. 6, Jul. 1999, pp.13-37.
3. Leveson, L., and Turner, C.S., *An Investigation of the Therac-25 Accidents*, IEEE Computer., 26, 7, (July 1993), pp.18-41.
4. Chung, L., B. A. Nixon, E. Yu and J. Mylopoulos, *Nonfunctional Requirements in Software Engineering*, Kluwer Academic Publishing, 2000.
5. Cysneiros, L.M., J.C.S. do Prado Leite, Non-functional Requirements: from Elicitation to Conceptual Models, IEEE Trans. On Soft Eng, 30(5), May, 2004, p.328-350.
6. Robinson, W., S. Pawlowski, and V. Volkov, Requirements Interaction Management, ACM Comput. Surv., 35 (2), pp.132–190, 2003.

7. Ryan A., An Approach to Quantitative Non-Functional Requirements in Software Development, Proc. the 34th Annual Government Electronics and Information Association Conference, 2000.
8. Kazman, R., Asundi, J., Klein, M.: Quantifying the Cost and Benefits of Architectural Decisions. In: Proc. Int. Conf. Software Eng. (2001) 297-306.
9. Azar, J., R. K. Smith, D. Cordes, Value Oriented Prioritization, IEEE Software, Jan, 2006.
10. Lehtola, L., M. Kauppinen, S. Kujala, Requirements Prioritization Challenges in Practice. Proc. of 5th Int´l Conf. On Product Focused Software Process Improvement (PROFES), Kansai Science City, Japan, April 2004, pp.497-508.
11. Berander, P., A. Andrews, Requirements Prioritization, in: A. Aurum, C. Wohlin (Eds.): Engineering and Managing Software Requirements, Springer, Berlin, Heidelberg, 2005, pp. 69-94.
12. Davis A., The Art of Requirements Triage. IEEE Computer, 36 (3), March, 2003, pp 42 – 49.
13. Larman C., "Applying UML and Patterns, *An Introduction to Object-Oriented Analysis and Design and the Unified Process*, 3rd edition. Upper Saddle River, NJ: Prentice Hall Inc., 2004.
14. Pfleeger, S. L., F. Wu, R. Lewis, *Software Cost Estimation and Sizing Methods*,: Issues and Guidelines, RAND Corporation, 2005.
15. Constantinides, C., and Skotiniotis, T., *Providing multidimensional decomposition in object-oriented analysis and design*, The IASTED International Conference on Software Engineering (SE 2004), February 17-19, 2004, Innsbruck, Austria.
16. Moreira A., Araujo J. and Brito I., *Crosscutting Quality Attributes for Requirements Engineering*, *In 14th Int. Conf. on Soft. Eng. and Knowledge Eng*. 2002. pp. 167-174, , Ischia, Italy, 2002.
17. Park D., Kang S., Lee J., *Design Phase Analysis of Software Performance Using Aspect-Oriented Programming*, In 5th Aspect-Oriented Modeling Workshop in Conjunction with UML 2004, Lisbon, Portugal, 2004.
18. Araujo, J., Moreira, A., Brito, I., and Rashid, A., *Aspect-Oriented Requirements With UML*, Workshop on Aspect-Oriented Modeling with UML (held with UML 2002).
19. Egyed, A. and P. Grunbacher, *Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help*, IEEE Software, November 2004: pp. 50-58.