Managing Conflicts among Non-Functional Requirements

Dewi Mairiza, Didar Zowghi, Nurie Nurmuliani
Faculty of Engineering and Information Technology
University of Technology Sydney, PO Box 123 Broadway, NSW 2007, Australia
mairiza@it.uts.edu.au; didar@it.uts.edu.au, nur@igreen.net

***Abstract*— Non-functional requirements (NFRs) tend to interfere, conflict, and contradict with one other. Unlike functional requirements, this inevitable conflict arises as a result of inherent contradiction among various types of NFRs. A number of techniques to deal with this conflict have been developed. Majority of them focus on categorizing, documenting, or listing the potential conflicts among NFRs. Several models that represent the positive or negative relationships among NFRs have also been published in literature. However, the interpretation of NFRs may vary depending on numerous factors, such as the context of the system being developed and stakeholder involvement. Consequently, the relationships among them are not always obvious. This paper investigates the gaps in the existing research literature about the conflicts among NFRs and proposes a framework to manage this type of conflict.**

Keywords- conflicts, non-functional requirements, relationship

I. INTRODUCTION

To develop a software product, the first thing to do is to understand what the system is supposed to do and how its utilization can support the goals of the individuals or businesses that will pay for that system [1]. A structured set of activities associated with understanding a product's necessary capabilities and attributes is known as requirements engineering [2]. Requirements Engineering (RE) is a sub-discipline of software engineering which consists of requirements development and requirements management [2] and covers all systematic activities in discovering, documenting, and maintaining a set of requirements for a computer-based system [3].

A number of studies report that failure to understand and manage requirements is one of the main reasons of software failure as well as increased project cost and schedule over-runs [4-6]. Many system failures are attributed to poor requirements analysis [7-10]. One of the well-known cases that is often discussed in the literature is the failure in London Ambulance System (LAS). LAS was immediately non-activated after deployment because it did not meet a type of software requirements, namely non-functional requirements. Failure of LAS system in dealing with several types of non-functional requirements indirectly caused fatality of the patients due to late medical treatment [11, 12].

Prior research reports that conflict is one of many characteristics of non-functional requirements [13]. Non-functional requirements tend to interfere, conflict or contradict with each other. Achieving a particular type of non-functional requirements can hurt the achievement of other type(s) of non-functional requirements as a result of inherent contradiction among them [13, 14]. The initial stage of a long term project of investigating conflicts among NFRs is described in this paper. It describes the state of the art on conflicts among non-functional requirements as well as identifying the gaps within the existing methods of managing these conflicts during software development. A preliminary model of proposed framework of managing the conflicts among NFRs is also presented.

This paper is organized in seven sections. The first section is introduction of non-functional requirements and their importance during the software development. The second section describes some essential concepts about NFRs – how software engineering community defines NFRs, characteristics and types of NFRs, and difficulty and challenge with NFRs. Essential concepts about conflicts among NFRs – the property, definition, and potential causes are described in section three, followed by the importance of dealing with conflicts among NFRs during software development project in section 4. The fifth section explains the methods to deal with conflicts among NFRs and the gap analysis for the opportunity of developing framework to manage conflicts among NFRs. A preliminary design of the framework is described in section 6. Then, this paper gives a conclusion and future work by highlighting some open issues which are acquired from literature analysis.

II. NON-FUNCTIONAL REQUIREMENTS

In the early eighties, the term non-functional requirements (NFRs) was introduced as the requirements that restrict the types of solution that the system might consider [15]. However, although the term NFRs has been in use for almost three decades, studies to date indicate that there is still no consensus in the software engineering community regarding the standard definition of NFRs. There is also little uniformity within RE research on defining, scoping and labelling NFRs.

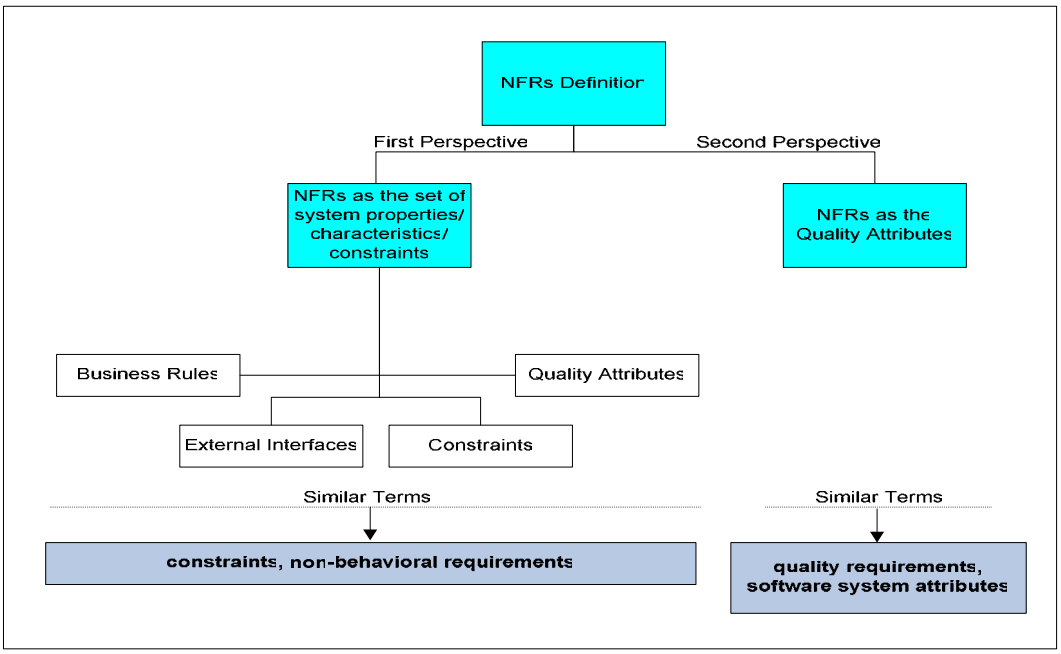

Figure 1 - Different Perspectives in Considering the Notion of NFRs

In literature, the term NFRs is considered for two different perspectives as shown in Figure 1: NFRs as the requirements that describe the properties, characteristics or constraints that a software system must exhibit; and NFRs as the requirements that describe the quality attributes that the software product must have. In the first perspective, NFRs consist of system constraints, business rules, external interfaces, quality attributes, and any other requirements that do not describe the functionality of the system. The second perspective only considers quality attributes as NFRs. For instance, maintainability, reusability, and usability are three types among huge number types of NFRs. For the purpose of this paper, NFRs is defined as the requirements that specify the desired quality attributes of the system being developed.

NFRs have several characteristics that make them different from FRs. NFRs are subjective, relative, and interacting [13]. Subjective because they can be viewed, interpreted, and evaluated differently by different people. Relative means the interpretation and importance of them may vary depending on the particular system being developed as well as the extent of stakeholders' involvement. Interacting because they tend to interfere, conflict or contradict with each other. This means that achieving a particular type of NFRs can hurt or help the achievement of other types of NFRs. Unlike FRs, NFRs are more abstract in nature [11, 16]. In software development, customers often state NFRs as general goals such as ease of use, the ability to recover from failure, or good response time. As a result, these vague goals leave the problem with different interpretation among stakeholder.

NFRs are not uniform in nature [17]. There are a large number of different types of NFRs. Each of types has different characteristics and roles during software development life cycle. For instance, security requirements describe various aspects about the security of the system such as authorization, privacy, and authentication while usability requirements describe various aspect of ease of use and user friendliness of the system. Due to these reasons, formally specifying NFRs is more difficult and complex than FRs [11, 16]. Our investigation on the types of NFRs discovers 248 types of NFRs exist in literature. Generally it can be classified as the quality attributes (e.g. maintainability, performance, and reliability); (development) constraints (e.g. timing, cost, and development personnel); interface requirements (e.g. user interface & human factors, look & feel, and system interfacing); and business rules (e.g. production life span). Further investigation to this superset list shows that only 106 types of NFRs correspond to the NFRs definition considered in this paper. Among them, 23 types (21.70%) have definition and characterization, 29 types (27.36%) only have definition, and 54 types (50.94%) were introduced without definition. The superset list of NFRs is illustrated in TABLE 1.

As described in the previous section, NFRs are important for the success of software project. However, in developing a software system, NFRs are often neglected, poorly understood and less considered. In the development of software system, users naturally focus on specifying their functional or behavioural requirements, i.e. the things the product must do [2, 13]. NFRs are often overlooked in the software development process [14, 18]. A number of studies investigating practices of dealing with NFRs in the software industry also report that commonly software developers do not pay sufficient attention to NFRs [14, 18-20]. NFRs are not elicited at the same time and the same level of details as the FRs [19, 20]. Additionally, NFRs are often poorly articulated in the requirements document [19, 20].

TABLE 1 - THE SUPERSET LIST OF NFRs TYPES

1. Accessibility/Access Control	37. Efficiency/Device Efficiency	72. Readability
2. Accountability	38. Enhanceability	73. Reconfigurability
3. Accuracy	39. Evolvability	74. Recoverability
4. Adaptability	40. Expandability	75. Reliability
5. Adjustability	41. Expressiveness	76. Repeatability
6. Affordability	42. Extendability	77. Replaceability
7. Agility	43. Extensibility	78. Replicability
8. Analyzability	44. Fault/Failure Tolerance	79. Reusability
9. Anonymity	45. Feasibility	80. Robustness
10. Attractiveness	46. Flexibility	81. Safety
11. Auditability	47. Functionality	82. Scalability
12. Augmentability	48. Generality	83. Security/Control and Security
13. Availability	49. Immunity	84. Self-Descriptiveness
14. Certainty	50. Installability	85. Simplicity
15. Changeability	51. Integratability	86. Stability
16. Communicativeness	52. Integrity	87. Standardizability/ Standardization/Standard
17. Compatibility	53. Interoperability	88. Structuredness
18. Completeness	54. Learnability	89. Suitability
19. Complexity/Interacting Complexity	55. Legibility	90. Supportability
20. Composability	56. Likeability	91. Survivability
21. Comprehensibility	57. Localizability	92. Susceptibility
22. Comprehensiveness	58. Maintainability	93. Sustainability
23. Conciseness	59. Manageability	94. Tailorability
24. Confidentiality	60. Maturity	95. Testability
25. Configurability	61. Measurability	96. Traceability
26. Conformance	62. Modifiability	97. Trainability
27. Consistency	63. Nomadicity	98. Transferability
28. Controlability	64. Observability	99. Trustability
29. Correctness	65. Operability	100. Understandability
30. Customizability	66. Performability	101. Usability
31. Debuggability	67. Performance/Efficiency/ Time or Space Bounds	102. Variability
32. Decomposability	68. Portability	103. Verifiability
33. Defensibility	69. Predictability	104. Viability
34. Demonstrability	70. Privacy	105. Visibility
35. Dependability	71. Provability	106. Wrappability
36. Effectiveness		

Research on non-functional requirements can be organized in various ways. In this paper, we have organized NFRs research into two main categories as illustrated in Figure 2. Process-oriented NFRs focuses on the process of engineering NFRs during the development of software system. The main research challenge is developing the appropriate methods to facilitate various activities in the engineering NFRs, such as how to elicit NFRs; how to model NFRs; and how to manage NFRs. Product-oriented NFRs focuses on the software system as the product of software development. The main research challenge is evaluating the software product to ascertain to which degree the product meets its NFRs. Typically, product-oriented NFRs research investigates various metrics and models to evaluate and measure NFRs. Managing conflicts among NFRs are considered under the managing NFRs category.

III. CONFLICTS AMONG NFRS

For many years, in various fields of knowledge such as sociology, psychology, politic or economy, it has been recognized that conflict is an inevitable attribute of interaction [21]. It is a common phenomenon that may arise in different contexts and levels [22]. In requirements engineering, the term "conflict" has been used to cover interference [21], inconsistency [23, 24], or interdependency [10, 25, 26] among requirements. Conflict among NFRs has been defined differently in literature and associated to the interacting characteristic of NFRs. From various definitions about conflicts among NFRs, it is revealed that the property of conflict among NFRs is interference – negative contribution of one NFR on another NFR [25]. This interference causes some tradeoffs in satisfying a set of

NFRs. As a result, a pair or a set of NFRs cannot be satisfied at the same time. In fact, in the research about conflicts among NFRs, researchers tend to focus on two different aspects of conflict: the relationships and the tradeoffs. Relationships focus on how NFRs contribute negatively to the other NFRs while the tradeoffs focus on the situation where we can not achieve two or more NFRs at the same time.

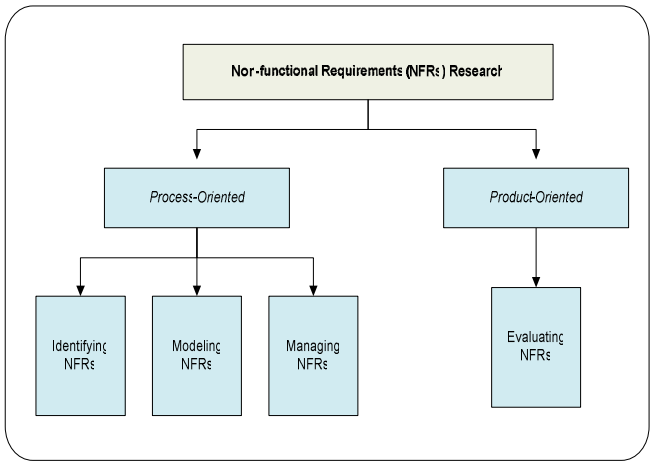

Figure 2 - NFRs Research

Generally there are two factors that potentially cause the conflict among NFRs. The first factor is different needs and perspectives among systems' stakeholders and the second factor is inherent contradiction among NFRs as the result of their specific characteristic – interacting character.

Like FRs, NFRs are also elicited from different stakeholder that often have different needs and perspectives over the system. Stakeholder will disagree over how to interpret features of the application domain, what the requirements for a new system are, and how to meet those requirements [21]. Different views of multiple stakeholders toward the system also cause various types of inconsistencies in software requirements, such as process-level deviation, instance-level deviation, terminology clash, and etc [27]. These differences may lead to inconsistency of requirements in the SRS document. Robinson, Pawlowski & Volkov [10] argue that inconsistent requirements often reflect the inconsistent needs of system stakeholders.

Unlike FRs, conflicts among NFRs suffer from severe tradeoffs among them as the result of their inherent contradiction [14]. Certain combination of NFRs in the software system may affect the inescapable trade offs [2, 14, 28]. For example, a requirement for a certain level of performance can be contradicted by security requirements which use processor capacity to carry out dynamic system checking [29]. This paper deals with the conflict among NFRs because of their inherent contradiction.

IV. The Importance of Dealing With Conflicts

Dealing with conflict among NFRs is important due to several reasons. The first reason is conflicts among software requirements are inevitable [13, 30-32]. Conflicting requirements are one of the three main problems in software development in term of the additional effort or mistakes attributed to them [32]. A multiple-project analysis study conducted by Egyed & Boehm [33, 34] in a period of two years reports that between 40% and 60% of requirements involved are in conflict, and among them, NFRs involved the greatest conflict, which was nearly half of requirements conflict [33 & 34 cited by 10].

Moreover, one of the characteristics of NFRs is interacting, which means NFRs tend to interfere, conflict or contradict with each other [13]. Certain combination of NFRs in the software system may affect the inescapable trade offs [2, 14, 28]. Therefore, managing conflict among NFRs as well as making these conflicts explicit are important [35]. Managing NFRs conflict is important for finding the right balance of non-functional requirements – a balance of attribute satisfaction – in achieving successful software products [2, 28].

Additionally, lessons learnt from practice confirms that one of the important aspects during NFRs specification is management of conflict among interacting NFRs [14]. Most systems experience severe tradeoffs among the major groups of NFRs, for example performance often interferes with maintainability and reusability. Experience from Alcatel shows that conflict resolutions for handling NFRs conflicts often results in changing overall design guidelines, not by simply changing one module [14].

V. Managing Conflicts Among NFRs

Generally, there are two main research challenges concerning conflict among NFRs: (1) to understand the nature of complex relationships among NFRs and (2) to develop techniques to deal with conflict among NFRs. In term of understanding the relationships, a number of potential conflict models have been presented in the literature. For instance, Wiegers [2] presents a matrix of positive and negative relationships among NFRs; Egyed & Grünbacher [36] present a model of potential conflict and cooperation; and Sadana & Liu [37] also present a model of relationship among ISO9126 quality attributes. These models illustrate some typical interrelationships among NFRs. In term of dealing with conflicts among NFRs, various techniques have been developed to manage this kind of conflict.

Managing conflict among NFRs consists of three main activities, conflict identification, conflict analysis, and conflict resolution. Conflict identification aims to detect the potential conflict. Conflict analysis aims to evaluate and investigate potential conflict and their tradeoffs. Conflict resolution aims to resolve the potential conflict. Various methods to identify, analyze, and resolve conflicts among NFRs are illustrated in TABLE 2. All of those methods were developed to address the following challenges:

- how to identify the conflicts early in the software development process
- how to identify the true conflicts and reduce false conflicts
- how to identify and capture the nature of conflicts

- how to identify, analyze and resolve the conflict automatically
- how to calculate the tradeoff among NFRs
- how to resolve the conflict
- how to facilitate the negotiation for conflict resolution

Further investigation to each method reveals that some methods are domain dependent [13, 38, 39] – it draws on domain knowledge to aid in assessing quality attributes [13, 40]. Other methods (e.g. [28]) are good for making top level suggestions about potential conflicts [41], but they lack detail for each identified conflict. They may also produce a significant number of potential conflicts since the catalogue provided does not support the true or false conflict identification. Another method covered in [36] is able to reduce and optimize the list of potential conflicts, but it performs late conflict identification. Detail analysis of each current method can be found in [42].

TABLE 2 - MANAGING CONFLICTS AMONG NFRs

Conflict Identification		
References	**Technique/Method**	**Objective**
[13]	catalogue of relationships among softgoals	to identify the conflicts among softgoals
[41]	examining quality-attribute tradeoffs in software architecture strategy	to identify conflicts early
[36]	model of potential conflict and cooperation	to identify the conflicts
[43]	modelling NFRs as constraint hierarchy	to identify the conflicts among NFRs based on their threshold values
[37]	potential conflict model of high level NFRs	to identify the conflicts

Conflict Analysis		
References	**Technique/Method**	**Objective**
[36]	requirements traceability (use trace analyzer to test cases and codes)	to analyse whether the potential conflicts detected are true conflicts or false conflicts in order to reduce false conflicts in the identification process
[43]	modeling NFRs as constraint hierarchy with attribute level of importance	to analyse the conflict among NFRs based on their level of importance
[44]	user preferences and NFR taxonomy	to analyse the tradeoffs among NFRs as the result of conflict and cooperation among NFRs
[37]	integrated analysis of FRs and NFRs	to decompose each conflicting NFRs based on the structure of relevance FRs and character of NFRs

Conflict Resolution		
References	**Technique/Method**	**Objective**
[28, 41]	negotiation	to facilitate negotiation
[45]	non-functional decomposition (NFD)	to resolve the conflict among NFRs
[43]	analysing NFRs constraint hierarchy and their attribute levels	to find the value of soft constraints that do not conflict with other constraints

Although conflict among NFRs has been acknowledged as one of the attributes of NFRs, and literature shows that managing this conflict is important, studies to date indicate that little progress has been made toward understanding such conflict, how this conflict arises during software development as well as how this conflict might be managed.

Majority of research on conflict among NFRs provide documentation, catalogue, or list of potential conflicts among the types of NFRs, which is known as potential conflict model. These potential conflict models are used to identify and analyze the conflict among NFRs during the development of software system. Apart from strength and weaknesses of each approach, however, NFRs are relative and subjective. The interpretation of NFRs may vary depending on the system being developed and the extent of stakeholder involvement [13]. NFRs can be defined, interpreted and characterized differently by different people and different context within which the system is being developed. Consequently, the positive or negative relationships among types of NFRs are not always obvious. These relationships might change depending on the meaning of NFRs in the context of the system being developed. Due to these relative and subjective characteristics of NFRs, cataloguing the NFRs relationships in order to represent the conflict among them would inevitably produce disagreement. Identifying the conflict among NFRs by using the catalogue of relationship among NFRs without understanding the meaning of NFRs in the system being developed may produce the erroneous conflict identification and analysis. Therefore, a technique to identify and reason with the conflict among NFRs by considering the relative characteristic of NFRs is needed.

Existing NFRs conflict identification techniques still fail to capture the nature of conflicts [37]. These techniques can only identify the conflicts in high level form, i.e. one type of NFR has the conflict with another type of NFR. The type, significance and the hierarchy of conflicts are still poorly understood. In fact, to perform the conflict resolution task, stakeholder must understand how the requirements affect one another [46]. Therefore, understanding the nature of conflict is also necessary to perform the task of conflict resolution.

Studies to date indicate that there is no systematic framework that allows developer to identify and analyze case by case in each system which NFRs of the system are in conflict and which NFRs are not. Such framework should be able to identify not only the existence of conflict, but also the type and significance of conflict and the appropriate potential strategy to resolve the conflict.

Existing potential conflict models that represent the relationship among NFRs are often in disagreement with each other. For example, according to Wiegers [2], efficiency has negative relationship with usability, but according to Egyed & Grünbacher [36], efficiency has positive relationship with usability. The disagreement in defining the relationship among NFRs is potentially influenced by several factors:

- Potential conflict model proposed by Egyed & Grünbacher [36] and Sadana & Liu [37] were developed from the literature analysis and not supported by empirical evidence.
- there is no general consensus in software engineering community regarding the definition and types of NFRs. Even, some NFRs were introduced without definition [47, 48]. Each type of NFRs can be defined and characterized in various ways. Diversity in defining and considering each type of NFRs may generate different meaning of NFRs and relationships that exist among them.
- the interpretation of NFRs may vary depending on the context of the system being developed and stakeholder involvement [13], and perhaps a number of other factors. As a result, the meaning of NFRs in each system is not always the same. Therefore, the relationships that exist among them are also not always obvious.

Literature review conducted also indicates that process-oriented NFRs research is more focused on developing various approaches for systematic treatment of NFRs. There is a lack of empirical research that investigates NFRs in practice, particularly the conflicts among them. How NFRs affect one another and how the conflict resolution is performed in practice is still not well understood. This lack of empirical study in understanding the multiple aspects of NFRs, including conflict among them is also reported by Paech & Kerkow [35]. Therefore, this research would investigate the conflicts among NFRs in order to increase our understanding of how NFRs affect one another and how this conflict might be managed.

In the field of software architecture, conflict among NFRs can be characterized as the consequence of candidate architecture towards multiple NFRs. Each design architecture has impacts on the ability of the system to meet its NFRs [49, 50]. The impacts may influence the likelihood of satisfaction of a particular type of NFRs. A number of architecture analysis methods that aim to evaluate the suitability of architecture to satisfy NFRs have been developed, for example SAAM [51], SAAMER [52], SBAR [53], ATAM [54], ABAS [55], and ALPSM [56]. However, majority of them only focus on individual NFRs in order to verify whether the candidate architecture satisfies the desired properties of an application. The interaction between multiple NFRs as a result of architecture decision is not elaborated. Only ATAM explicitly evaluates the software architecture's fitness with respect to multiple NFRs, therefore tradeoffs among NFRs as the consequence of design decision is identifiable. However, none of the methods investigate the conflict within NFRs. Architecture analysis methods only provide a warning when the selected architecture produces the tradeoffs against multiple NFRs.

VI. PROPOSED FRAMEWORK

According to the analysis that we have conducted in the previous section, in this section we describes a preliminary framework for managing the conflicts among NFRs by considering the relevant characteristic of NFRs. Such framework should be able to identify not only the existence

of conflict, but also the type and significance of conflict, as well as the appropriate potential strategy to resolve the conflict (e.g. business prioritization). This framework consists of a technique to identify the conflicts, taxonomy to characterize the conflicts, and a catalogue of strategies to resolve these conflicts as illustrated in Figure 3.

By providing a mapping between all of these components, this framework should be able to identify which NFRs of the system are in conflicts, what type of conflicts and how significance this conflict, and which associated potential strategies to resolve these conflicts. Thus, by executing NFRs of the system as the input, through this proposed framework, we can identify a set of NFRs of the system that are in conflict, type and significance of the identified conflict, and recommendation of potential strategies to resolve the conflict as the output.

VII. CONCLUSION

This paper describes the state of the art of non-functional requirements research, particularly about conflicts among non-functional requirements. Various perspectives of software engineering community in considering NFRs and the contradiction between the importance of NFRs for the success of software project and the practices of dealing with NFRs in developing a software system have also been discussed. All types of NFRs extracted by conducting content analysis on literature have also been illustrated. The literature review about the importance of dealing with conflicts among NFRs, potential causes of conflicts, and the property of conflicts among NFRs have also been explained. Several methods to manage the conflicts among NFRs have been briefly described as well as the gaps found from these methods. From these gaps, a preliminary framework is proposed to comprehensively deal with conflicts among NFRs. This proposed framework should be able to identify not only the existence of conflict, but also the type and significance of conflict, as well as the appropriate potential strategy to resolve the conflict.

Next research will focus on continuing the development of this preliminary framework. Some data collection and analysis will be conducted in order to identify the existence of conflict as well as to characterize the conflict. Mapping between the taxonomy of conflict and the conflict resolution strategy will also be established.

ACKNOWLEDGMENT

We would like to thank The International Schlumberger Foundation Paris for funding this research through Faculty for the Future Award Program.

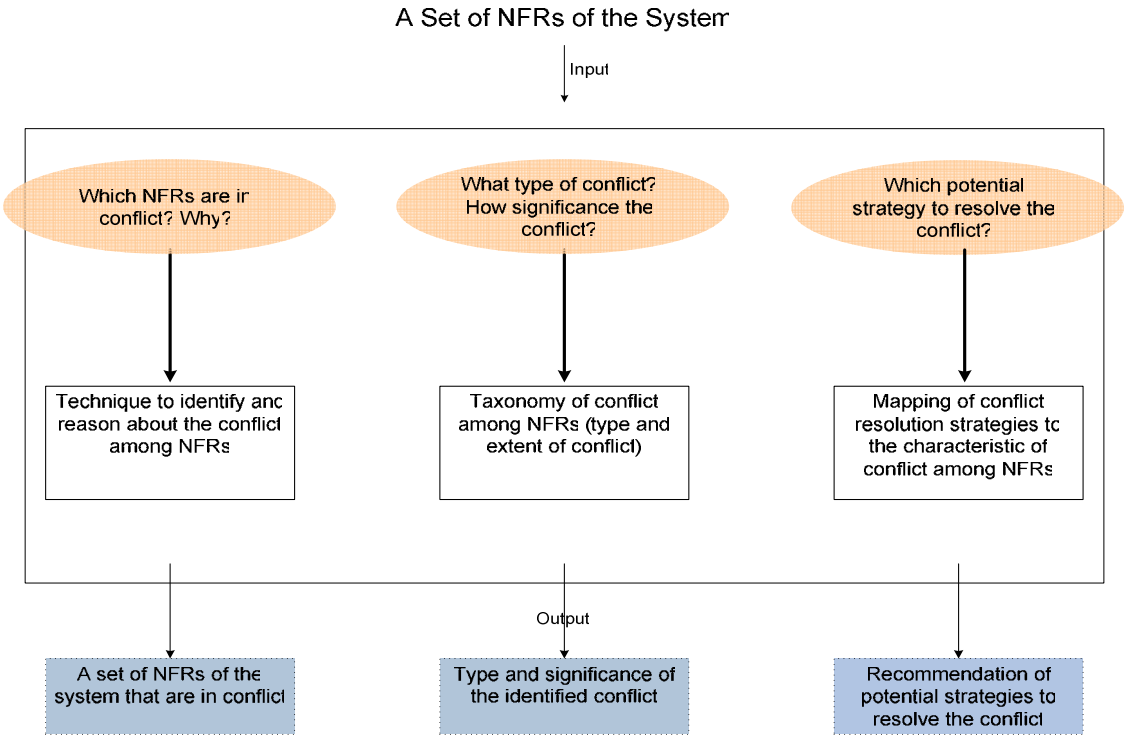

Figure 3 - Framework to Manage Conflicts among NFRs

REFERENCES

[1] I. Sommerville, "Integrated requirements engineering: a tutorial," *IEEE Software,* pp. 16-23, 2005.

[2] K. E. Wiegers, *Software requirements*, 2nd ed. Washington: Microsoft Press, 2003.

[3] I. Sommerville and P. Sawyer, *Requirements engineering: a good practice guide*. Chicester, England.: John Wiley & Sons Ltd, 1997.

[4] "Chaos," The Standish Group 1995.

[5] "Extreme chaos," The Standish Group International, Inc 2001.

[6] D. Zowghi and N. Nurmuliani, "A study of the impact of requirements volatility on software project performance," in *APSEC 02*: IEEE, 2002.

[7] C. Jones, *Patterns of software systems failure and success*. London, UK: International Thomson Computer Press, 1996.

[8] K. Lyytinen and R. Hirschheim, *Information systems failures—a survey and classification of the empirical literature*: Oxford University Press, Inc., 1988.

[9] P. G. Neumann, *Computer related risks*. Reading, MA: Addison-Wesley, 1995.

[10] W. N. Robinson, S. D. Pawlowski, and V. Volkov, "Requirements interaction management," *ACM Computing Surveys,* vol. 35, pp. 132-190, 2003.

[11] K. K. Breitman, J. C. S. Prado Leite, and A. Finkelstein, "The world's a stage: a survey on requirements engineering using a real-life case study," *Journal of the Brazilian Computer Society,* vol. 6, pp. 1-57, 1999.

[12] A. Finkelstein and J. Dowell, "A comedy of errors: the London ambulance service case study," in *Eigth International Workshop Software Specification and Design*, 1996, pp. 2-5.

[13] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*. Massachusetts: Kluwer Academic Publishers, 2000.

[14] C. Ebert, "Putting requirement management into praxis: dealing with nonfunctional requirements," *Information and Software Technology,* vol. 40, pp. 175-185, 1998.

[15] R. T. Yeh, "Requirements analysis - a management perspective," in *COMPSAC '82*, 1982, pp. 410-416.

[16] G.-C. Roman, "A taxonomy of current issues in requirements engineering," *Computer,* vol. 18, pp. 14 - 23, 1985.

[17] D. Firesmith, "Using quality models to engineer quality requirements," *Journal of Object Technology,* vol. 2, pp. 67-75, 2003.

[18] D. J. Grimshaw and G. W. Draper, "Non-functional requirements analysis: deficiencies in structured methods," *Information and Software Technology,* vol. 43, pp. 629-634, 2001.

[19] N. Heumesser, A. Trendowicz, D. Kerkow, H. Gross, and L. Loomans, "Essential and requisites for the management of evolution - requirements and incremental validation," Information Technology for European Advancement, ITEA-EMPRESS consortium 2003.

[20] N. Yusop, D. Zowghi, and D. Lowe, "The impacts of non-functional requirements in web system projects," *International Journal of Value Chain Management* vol. 2, pp. 18-32, 2008.

[21] S. Easterbrook, "Handling conflict between domain descriptions with computer-supported negotiation," *International Journal of Knowledge Acquisition,* vol. 3, pp. 255-289, 1991.

[22] S. Byrne and J. Senehi, *Conflict analysis and resolution as a multidiscipline; a work in progress*. New York, USA: Routledge, 2009.

[23] W. N. Robinson and V. Volkov, *Requirements conflict restructuring*. Atlanta: GSU CIS Working Paper, Georgia State University, 1999.

[24] S. Easterbrook and B. Nuseibeh, "Using viewpoints for inconsistency management," *Software Engineering Journal,* vol. 11, pp. 31-43, 1996.

[25] Moreira A., Rashid A., and J. Araujo, "Multi-dimensional separation of concerns in requirements engineering," in *13th IEEE International Conference on Requirements Engineering (RE '05)*, 2005.

[26] M. Kim, S. Park, V. Sugumaran, and H. Yang, "Managing requirements conflicts in software product lines: a goal and scenario based approach," *Data & Knowledge Engineering,* vol. 61, pp. 417-432, 2007.

[27] A. Lamsweerde, R. Darimont, and E. Letier, "Managing conflicts in goal-driven requirements engineering," *IEEE Transactions on Software Engineering Special Issue on Inconsistency Management in Software Development,* November 1998 1998.

[28] B. Boehm and H. In, "Identifying quality-requirements conflict," *IEEE Software,* vol. 13, pp. 25-35, 1996.

[29] G. Kotonya and I. Sommerville, *Non-functional requirements*, 1998.

[30] L. Chung, B. A. Nixon, and E. Yu, "Using non-functional requirements to systematically support change," in *The second international symposium on requirements engineering*, York, 1995, pp. 132-139.

[31] L. Chung, B. A. Nixon, and E. Yu, "Dealing with change: an approach using non-functional requirements," *Requirements Engineering,* vol. 1, pp. 238-260, 1996.

[32] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Communication of the ACM,* vol. 31, pp. 1268-1287, 1988.

[33] A. Egyed and B. Boehm, "A comparison study in software requirements negotiation," in *8th Annual International Symposium on Systems Engineering (INCOSE'98)*, 1998.

[34] B. Boehm and A. Egyed, "WinWin requirements negotiation processes: a multi-project analysis," in *5th International Conference on Software Processes*, 1998.

[35] B. Paech and D. Kerkow, "Non-functional requirements engineering - quality is essential," in *10th International Workshop on Requirements Engineering: Foundation for Software Quality*, 2004, pp. 27-40.

[36] A. Egyed and P. Grünbacher, "Identifying requirements conflicts and cooperation: how quality attributes and automated traceability can help," *IEEE Software,* vol. 21, pp. 50 - 58, 2004.

[37] V. Sadana and X. F. Liu, "Analysis of conflict among non-functional requirements using integrated analysis of functional and non-functional requirements," in *31st International Computer Software and Applications Conference (COMPSAC 2007)*, 2007.

[38] J. Mylopoulos, L. Chung, and B. A. Nixon, "Representing and using nonfunctional requirements: a process-oriented approach," *IEEE Transaction on Software Engineering,* vol. 18, 6 June 1992.

[39] E. S. K. Yu and J. Mylopoulos, "An actor dependency model of organizational work: with application to business process reengineering," in *Conference on Organizational Computing Systems (COOCS'93)*, Milpitas, CA, 1993.

[40] B. Boehm and H. In, "Aids for identifying conflicts among quality requirements," *IEEE Software, March 1996,* 1996.

[41] H. In, B. Boehm, T. Rodgers, and M. Deutsch, "Aplying WinWin to quality requirements: a case study," in *23rd International Conference on Software Engineering*, Toronto, Ontario, Canada, 2001, pp. 555 - 564.

[42] D. Mairiza, "Doctoral Assessment Report: Investigating conflicts among non-functional requirements," University of Technology Sydney, Sydney 2009.

[43] Y. Guan and A. K. Ghose, "Use constraint hierarchy for non-functional requirements analysis," *Lecture Notes in Computer Science,* vol. 3579/2005, pp. 104-109, 2005.

[44] G. S. A. Mala and G. V. Uma, "Elicitation of non-functional requirements preference for actors of usecase from domain model," *Lecture Notes in Computer Science,* vol. 4303/2006, pp. 238-243, 2006.

[45] E. R. Poort and P. H. N. de With, "Resolving requirement conflicts through non-functional decomposition," in *Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA '04)*, 2004.

[46] D. G. PracticeTM, *Requirements trade-offs/negotiations*: Gold PracticeTM, 2004.

[47] M. Glinz, "Rethinking the notion of non-functional requirements," in *Third World Congress for Software Quality*, Munich, Germany, 2005, pp. 55-64.

[48] M. Glinz, "On non-functional requirements," in *15th IEEE International Requirements Engineering Conference (RE '07)*, 2007, pp. 21-26.

[49] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method," in *4th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '98)*, 1998, pp. 68-78.

[50] J. Bosch and P. Molin, "Software architecture design: evaluation and transformation," in *IEEE Conference and Workshop on Engineering of Computer-Based Systems*, 1999, pp. 4-10.

[51] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-based analysis of software architecture," *IEEE Software,* vol. 13, pp. 47-55, 1996.

[52] L. Chung, S. Bot, K. Kalaichelvan, and R. Kazman, "An approach to software architecture analysis for evolution and reusability," in *1997 Conference of the Centre for Advanced Studies on Collaborative Research*, Toronto, Canada, 1997, p. 15.

[53] P. Bengtsson and J. Bosch, "Scenario-based software architecture reengineering," in *5th International Conference on Software Reuse*, 1998, pp. 308-317.

[54] R. Kazman, M. Klein, and P. Clements, "ATAM: method for architecture evaluation," Software Engineering Institute - Carnegie Mellon University 2000.

[55] M. Klein and R. Kazman, "Attribute-based architectural styles," Software Engineering Institute - Carnegie Mellon University 1999.

[56] P. Bengtsson and J. Bosch, "Architecture level prediction of software maintenance," in *3rd European Conference on Software Maintenance and Reengineering*, 1999, pp. 139-147.