Constructing a Catalogue of Conflicts among Non-functional Requirements

Dewi Mairiza and Didar Zowghi

School of Software, Faculty of Engineering and Information Technology,
University of Technology, Sydney (UTS), Australia
{Dewi.Mairiza,Didar.Zowghi}@uts.edu.au

Abstract. Non-Functional Requirements (NFRs) are recognized as a critical factor to the success of software projects because they address the essential issue of software quality. NFRs tend to interfere, conflict, and contradict with one another and this conflict is widely acknowledged as one of the key characteristics of NFRs. Several models of NFRs conflicts have been proposed and the interacting nature of NFRs has been characterized as either positive or negative inter-relationships among NFRs. Positive relationship represents a pair of NFRs that are supporting each other while negative relationship represents those NFRs that are conflicting with one another. Furthermore, as NFRs are also relative, the interpretation of NFRs may vary depending on many factors such as the context of the system being developed and the extent of stakeholders' involvement. The multiple interpretations of NFRs may lead to positive or negative inter-relationships that are not always obvious. These relationships may change depending on the meaning of NFRs in the system being developed. Hence, the existing potential conflicts models remain in disagreement with one other. This paper presents the result of an extensive and systematic investigation of the extant literature over the notion of NFRs and the conflicts among them. Rigorous synthesis of the carefully reviewed literature has resulted in the construction of a catalogue of NFRs conflicts with respect to NFRs relative characteristic. The relativity of conflicts is characterized by three categories: *absolute conflict; relative conflict; and never conflict.* This comprehensive catalogue could assist software developers with identifying the NFRs conflicts, performing conflicts analysis, and suggesting potential strategies to resolve these conflicts.

Keywords: Non-functional requirements, Relationship, Conflict, Relative, Catalogue.

1 Introduction

In the early eighties, the term Non-Functional Requirements (NFRs) was introduced as those requirements that restrict the type of solutions that a software system might consider [1]. However, although this term has been in use for almost three decades, studies to date indicate that currently there is no general consensus in the software or systems engineering community regarding the notion of NFRs. In the literature, the term NFRs is considered within two different perspectives: (1) NFRs as the requirements that describe the properties, characteristics or constraints that a software system must exhibit; and (2) NFRs as the requirements that describe the quality attributes that the software product must have [2].

In software development, NFRs are recognized as a critical factor to the success of software projects. NFRs address the essential issue of the quality of the system [3-5]. Without well-defined NFRs, a number of potential problems may occur, such as a software which is inconsistent and of poor quality; dissatisfaction of clients, end-users, and developers toward the software; and causing time and cost overrun for fixing the software [5]. NFRs are also considered as the constraints or qualifications of the operations [6]. They place restrictions on the product being developed, development process, and specify external constraints that the product must exhibit [7]. Charette [8] claims that NFRs are often more critical than individual Functional Requirements (FRs) in the determination of a system's perceived success or failure [9, 10]. Neglecting NFRs has led to a series of software failures. For example systemic failure in London Ambulance System [11, 12], performance and scalability failure in the New Jersey Department of Motor Vehicles Licensing System [13], failure in the initial design of the ARPANet Interface Message Processor Software [14], and some other examples as described in [11, 13-15].

Although NFRs are widely recognized to be very significant in the software development, a number of empirical studies reveal that NFRs are often neglected, poorly understood and not considered adequately in developing the software applications. In the development of software systems, users naturally focus on specifying their functional or behavioral requirements, i.e. the things the product must do [5, 9]. NFRs are often overlooked in the software development process [3, 16]. A number of studies investigating practices of dealing with NFRs in the software industry also reported that commonly software developers do not pay sufficient attention to NFRs [3, 16-18]. NFRs are not elicited at the same time and the same level of details as the FRs and they are often poorly articulated in the requirements documents [17, 18]. Furthermore, in the requirements engineering literature, NFRs have received less attention and not as well understood as FRs [5]. Majority of software engineering research, particularly within requirements engineering area only deal with FRs, i.e. ensuring that the necessary functionality of the system is delivered to the user [19]. Consequently, capturing, specifying, and managing NFRs are still difficult to perform due to most of software developers do not have adequate knowledge about NFRs and little help is available in the literature [20].

NFRs tend to interfere, conflict, and contradict with one another. Unlike FRs, this inevitable conflict arises as a result of inherent contradiction among various types of NFRs [3, 5]. Certain combinations of NFRs in the software system may affect the inescapable trade offs [3, 9, 13]. Achieving a particular type of NFRs can hurt the achievement of the other type(s) of NFRs. Hence, this conflict is widely acknowledged as one of many characteristics of NFRs [5].

Prior studies reveal that dealing with NFRs conflicts is essential due to several reasons [2]. First of all, conflicts among software requirements are inevitable [5, 21-23]. Conflicting requirements are one of the three main problems in software development in term of the additional effort or mistakes attributed to them [23]. A study of two-year multiple-project analysis conducted by Egyed & Boehm [24, 25] 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 conflicts [26]. Lessons learnt from industrial practices also confirm that one of the essential aspects during NFRs specification is management of conflicts among interacting NFRs [3]. Experience shows most systems suffer with severe tradeoffs among the major groups of NFRs. For example: the tradeoffs between security and performance requirements; or between security and usability requirements. In fact, conflicts resolutions for handling NFRs conflicts often result in changing overall design guidelines, not by simply changing one module [3]. Therefore, since conflicts among NFRs have also been widely acknowledged as one of NFRs characteristics, managing these conflicts as well as making them explicit is essential [19]. NFRs conflicts management is important for finding the right balance of attributes satisfaction, in achieving successful software products [9, 13].

A review of various techniques to manage the conflicts among NFRs have been presented in the literature [2]. Majority of these techniques provide a documentation, catalogue, or list of potential conflicts. These catalogues represent the interrelationships among various types of NFRs. Apart from strength and weaknesses of each technique, however, NFRs are also relative [5]. This means that the interpretation and importance of NFRs may vary depending on many factors, such as the particular system being developed as well as the extent of stakeholder involvement. NFRs can be viewed, interpreted, and evaluated differently by different people and different contexts within which the system is being developed. Consequently, the positive or negative relationships among them are not always obvious. These relationships might change depending on the meaning of NFRs in the context of the system being developed. Due to this relative characteristic of NFRs, existing potential conflicts models that represent the relationship among NFRs are often in disagreement with each other. For example, according to Wiegers [9] efficiency requirements have negative relationship (conflict) with usability requirements, but according to Egyed & Grünbacher [27] these two types of NFRs have positive relationship (support). Given that none of the existing conflicts catalogues deal with the relative characteristics of NFRs, we are motivated to pose the following research question:

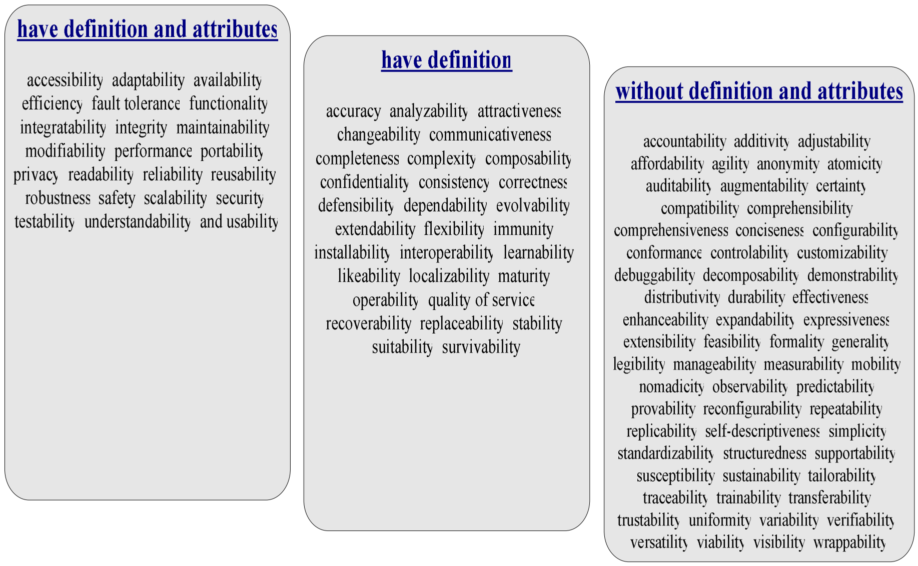
“Can a catalogue of conflicts among NFRs be developed with respect to the relative characteristic of NFRs?”

The catalogue of conflicts with respect to the NFRs relative characteristic that has been developed from a rigorous synthesis of the literature from several disciplines is presented as the novel contribution of this paper. This catalogue is built as a two-dimensional matrix that represents the conflict-relationships between various types of NFRs, i.e. how each type of NFRs is associated with the other types of NFRs considering the NFRs relative characteristic. The conflict-relationships are represented in three categories: *absolute conflict; relative conflict; and never conflict.*

This article is organized in six sections. The first section is the introduction to NFRs and conflicts among them. The second section describes the research framework and source of information used in this study. The superset list of NFRs is presented in section three continued by presenting the catalogue of NFRs conflicts in section four. Section five describes the benefits and potential applications of the conflicts catalogue in the software development projects. Then, section six concludes this paper by highlighting some open issues that are acquired from the investigation.

2 Catalogue Framework

To get a significant and comprehensive snapshot of the NFRs conflicts model, an extensive investigation of the literature over the last three decades has been performed. This investigation was conducted by exploring the articles from academic resources and documents from software development industry. Four general types of sources of information have been identified: (1) journal papers; (2) conference proceedings; (3) books; and (4) documents from software industry. Selection of those sources is made in order to confirm the completeness of the information by obtaining the academics and practitioners perspectives related to the notion of NFRs and conflicts among them. The study conducted by Chung et al. [5] was used as the starting point for selection of the papers to be reviewed.



have definition and attributes

accessibility adaptability availability efficiency fault tolerance functionality integratability integrity maintainability modifiability performance portability privacy readability reliability reusability robustness safety scalability security testability understandability and usability

have definition

accuracy analyzability attractiveness changeability communicativeness completeness complexity composability confidentiality consistency correctness defensibility dependability evolvability extendability flexibility immunity installability interoperability learnability likeability localizability maturity operability quality of service recoverability replaceability stability suitability survivability

without definition and attributes

accountability additivity adjustability affordability agility anonymity atomicity auditability augmentability certainty compatibility comprehensibility comprehensiveness conciseness configurability conformance controlability customizability debuggability decomposability demonstrability distributivity durability effectiveness enhanceability expandability expressiveness extensibility feasibility formality generality legibility manageability measurability mobility nomadicity observability predictability provability reconfigurability repeatability replicability self-descriptiveness simplicity standardizability structuredness supportability susceptibility sustainability tailorability traceability trainability transferability trustability uniformity variability verifiability versatility viability visibility wrappability

Fig. 1. NFRs Types in the Literature

Our study has examined 182 sources of information. All of them are literatures within the discipline of software engineering. They cover various issues of NFRs and conflicts among them. The research articles reviewed are published in key journals and conference proceedings of the software engineering literature, such as the Journal of Systems and Software; IEEE Transaction on Software Engineering; IEEE Software; Lecture Notes in Computer Science; Journal of Information and Software Technology; Requirements Engineering Journal; Requirements Engineering Conference, International Conference on Software Engineering, and Requirements Engineering Foundations of Software Quality Workshop.

Each source was then systematically analyzed using content analysis technique. Content analysis is a research technique that uses a set of procedures to make valid inferences from texts or other meaningful matter [28, 29]. This technique is well founded and has been in used for over sixty years. The analysis covers three essential issues: the NFRs types, the definition and attributes[1] of each type, and the conflict interdependencies among them. Content analysis technique was selected because it enables researchers to identify trends and patterns in the literature through the frequency of keywords, and by coding and categorizing the data into a group of words with similar meaning or connotations [29, 30]. Furthermore, this technique is also applicable to all domain contexts [28, 31].

To develop a catalogue of NFRs conflicts, a research framework was followed. This framework consists of three research stages:

(1) to create a comprehensive catalogue of NFRs types, their definition and attributes characterization
(2) to identify the interdependencies among NFRs
(3) to perform a normalization process to standardize the NFRs in the conflicts catalogue

Since there is no standard catalogue of NFRs types available in the literature and previous studies [32-34] also claimed that many types of NFRs were introduced without definition or attributes characterization, the first stage of the research was creating a comprehensive catalogue of NFRs types. Each type of NFRs discussed in the literature was recorded. The definitions and attributes correspond to each of NFRs type were also documented. Conflicting terminologies and definitions were handled through the frequency analysis technique and keywords identification.

Table 1. NFRs Types in the Initial Catalogue

NFRs Types		
Accuracy	Interoperability	Reliability
Analyzability	Legibility	Reusability
Availability	Maintainability	Robustness
Compatibility	Performance	Safety
Confidentiality	Portability	Security
Dependability	Privacy	Testability
Expresiveness	Provability	Understandability
Flexibility	Recoverability	Usability
Functionality		Verifiability

The second stage of the research was creating an initial catalogue of the conflicts among NFRs. In this stage, NFRs conflict relationships were used as the criteria to develop the catalogue. This stage was initiated by identifying the interdependencies

[1] In this paper, the term attribute is considered as the major components of each NFRs type. In the literature, attribute is also referred as NFRs subtype [5] or quality sub factors [4].

among various types of NFRs. These interdependencies represent the typical interrelationships of a particular type of NFRs towards another type of NFRs (e.g. positive, negative, or neutral interrelationships). This investigation produced the initial catalogue that presents the conflict relationships among 26 types of NFRs. These NFRs types are listed in Table 1.

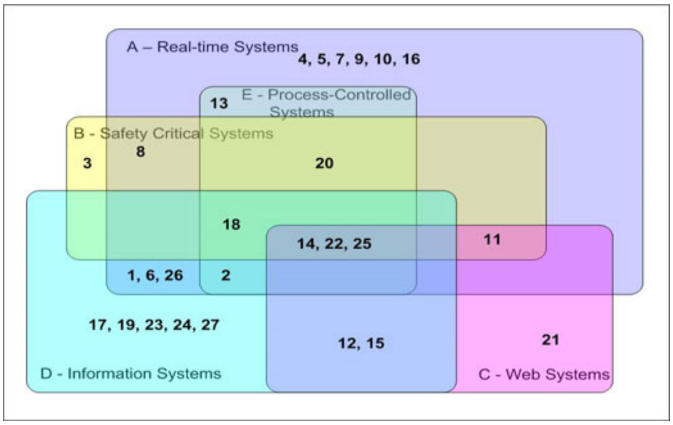
Table 2. NFRs Definition and Attributes [34]

NFRs	Definition	Attributes
Performance	requirements that specify the capability of software product to provide appropriate performance relative to the amount of resources needed to perform full functionality under stated conditions	response time, space, capacity, latency, throughput, computation, execution speed, transit delay, workload, resource utilization, memory usage, accuracy, efficiency compliance, modes, delay, miss rates, data loss, concurrent transaction processing
Reliability	requirements that specify the capability of software product to operates without failure and maintains a specified level of performance when used under specified normal conditions during a given time period	completeness, accuracy, consistency, availability, integrity, correctness, maturity, fault tolerance, recoverability, reliability, compliance, failure rate/critical failure
Usability	requirements that specify the end-user-interactions with the system and the effort required to learn, operate, prepare input, and interpret the output of the system	learnability, understandability, operability, attractiveness, usability compliance, ease of use, human engineering, user friendliness, memorability, efficiency, user productivity, usefulness, likeability, user reaction time
Security	requirements that concern about preventing unauthorized access to the system, programs, and data	confidentiality, integrity, availability, access control, authentication
Maintainability	requirements that describe the capability of the software product to be modified that may include correcting a defect or make an improvement or change in the software	testability, understandability, modifiability, analyzability, changeability, stability, maintainability compliance

The next stage was performing a normalization process against 26 types of NFRs that have been identified in the initial catalogue. This normalization was conducted in order to standardize the data obtained in the previous stage. Normalization is the process of removing the irrelevant NFRs, i.e. the types of NFRs that do not have definition and/or attributes, from the initial catalogue. The objective is to produce a conflicts catalogue of the well-defined NFRs types. In this normalization, the catalogue of NFRs types, their definitions, and their attributes are utilized as the basis of removing those irrelevant NFRs. This process has removed six NFRs from the initial catalogue. They are *compatibility, expressiveness, legibility, provability, verifiability* and *analyzability*. Therefore, the final conflicts catalogue is a two-dimensional matrix that represents the conflict interrelationships among 20 types of "normalized" NFRs.

3 NFRs Types

Various authors (e.g. [5, 35, 36]) define the term NFRs as the requirements that specify the desired quality attributes of the system. According to this definition, our analysis of NFRs types in the literature has resulted in identification of 114 types of NFRs. The superset list of these 114 NFRs types can be found in our previous publication [34].



Legend:

1	Accuracy	10	Installability	19	Reusability
2	Availability	11	Integrity	20	Safety
3	Communicativeness	12	Interoperability	21	Scalability
4	Compatibility	13	Maintainability	22	Security
5	Completeness	14	Performance	23	Standardizability
6	Confidentiality	15	Privacy	24	Traceability
7	Conformance	16	Portability	25	Usability
8	Dependability	17	Provability	26	Verifiability
9	Extensibility	18	Reliability	27	Viability

Fig. 2. Mapping of Concerned NFRs and Types of Systems [34]

Further investigation to the superset list indicates that 23 NFRs types (20.18%) have definition and attributes, 30 types (26.32%) only have definition, and the rest 61 types (53.50%) were introduced without definition or attributes. Since this finding indicates that more than 50% of NFRs listed in the literature do not have any definitions and attributes characterization, therefore, it confirms the previous claim made by Glinz [32, 33] that stated that "in the literature, many NFRs were introduced without definition or clarifying examples". The detailed list of this classification is presented in Fig. 1. In addition, the top five of the most frequently discussed NFRs types in the literature are presented in Table 2 and the concerned NFRs in various types of systems are presented in Fig. 2.

4 Catalogue of Conflicts

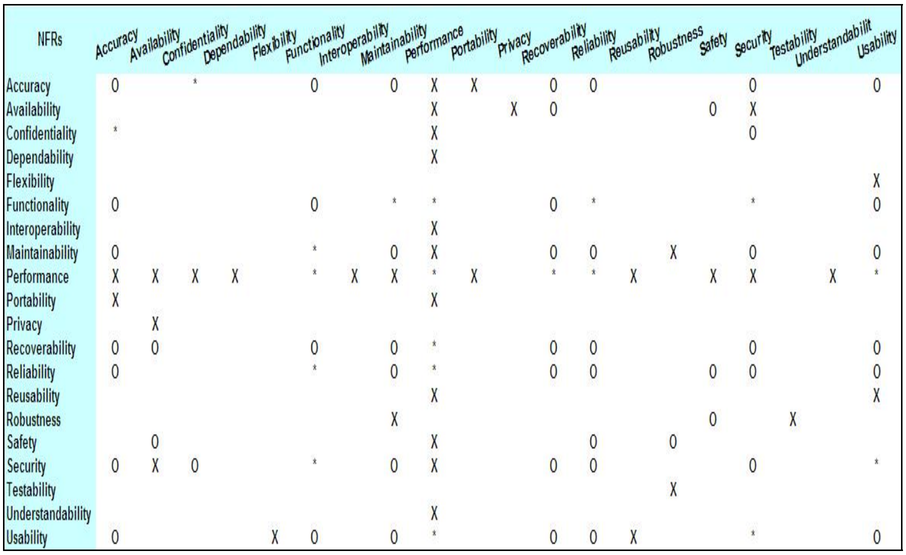
The catalogue of conflicts is a two-dimensional matrix that represents the typical interrelationships among 20 types of normalized NFRs, in term of the conflicts emerge among them. In this catalogue, the relativity of NFRs conflicts is presented in three categories: *absolute conflict; relative conflict;* and *never conflict* (as presented in Fig. 3).

- *absolute conflict.* this relationship represents a pair of NFRs types that are always in conflict. In the catalogue, this conflict relationship is labeled as 'X'.
- *relative conflict.* this relationship represents a pair of NFRs types that are sometimes in conflict. It consists of all pairs of NFRs that are claimed to be in conflict in a certain case but they are also claimed as not being in conflict in the other cases. This disagreement occurs due to several factors, such as the different interpretation/meaning of NFRs in the system being developed, the context of the system, the stakeholders' involvement, and the architectural design strategy implemented in that system. In the conflicts catalogue, this type of conflict relationship is labeled as '*'.
- *never conflict.* this relationship represents a pair of NFRs types that in the software development projects are never in conflict. It consists of all pairs of NFRs who have never been declared as being in conflict with each other. They may contribute either positively (e.g. support [37] or cooperative [27]) or indifferent to one another (e.g. low or very little impact on the other [9]).

Further analysis of the conflicts catalogue indicates that 36 pairs of NFRs are absolute conflict (e.g. accuracy and performance; security and performance; and usability and reusability); 19 pairs are relative conflict (e.g. reliability and performance; usability and security; and performance and usability); and 50 pairs are never conflict (e.g. accuracy and maintainability; security and accuracy; and usability and recoverability). The rest of relationships are not known due to there is no information available in the literature about how those pairs of NFRs contribute to each other. In the conflicts catalogue, this unknown conflict is presented as "the blank spaces".

Furthermore, this catalogue shows that NFRs with the most conflict with other NFRs is performance. Performance has absolute conflict with accuracy, availability, confidentiality, dependability, interoperability, maintainability, portability, reusability, safety, security, and understandability, and it has relative conflict with functionality, recoverability, reliability, and usability.

The investigation also indicates that certain attributes of a particular type of NFR can be in conflict with each other. This conflict points to the self-conflicting relationships for a particular type of NFR. Self-conflicting relationship is defined as a situation where the attributes of a single type of NFRs are in conflict. One of the examples is the relative conflict between performance and performance requirements. Performance requirements can be characterized among others by "response time" and "capacity". In many systems, these two attributes are in conflict. For example in a road traffic pricing system [38, 39], multi-user attribute[2] has negative contribution to the response time of the system. This means that increasing the number of concurrent users in the system may diminish the response time of the system.



NFRs	Accuracy	Availability	Confidentiality	Dependability	Flexibility	Functionality	Interoperability	Maintainability	Performance	Portability	Privacy	Recoverability	Reliability	Reusability	Robustness	Safety	Security	Testability	Understandabilit	Usability
Accuracy	0		*			0		0	X	X		0	0				0			0
Availability									X		X	0				0	X			
Confidentiality	*								X								0			
Dependability									X											
Flexibility																				X
Functionality	0					0		*	*			0	*				*			0
Interoperability									X											
Maintainability	0					*		0	X			0	0		X		0			0
Performance	X	X	X	X		*	X	X	*	X		*	*	X		X	X		X	*
Portability	X								X											
Privacy		X																		
Recoverability	0	0				0		0	*			0	0				0			0
Reliability	0					*		0	*			0	0			0	0			0
Reusability									X											X
Robustness								X								0		X		
Safety		0							X				0		0					
Security	0	X	0			*		0	X			0	0				0			*
Testability															X					
Understandability									X											
Usability	0				X	0		0	*			0	0	X			*			0

Fig. 3. Catalogue of Conflicts Among NFRs

Table 3. Conflicting NFRs in Literature

Conflicting NFRs	Nature of Conflict	%
Security and Performance	absolute	33%
Security and Usability	relative	23%
Availability and Performance	absolute	20%
Performance and Portability	absolute	17%
Reusability and Performance	absolute	17%
Interoperability and Performance	absolute	10%
Maintainability and Performance	absolute	10%
Reliability and Performance	relative	10%
Usability and Performance	relative	10%
Usability and Reusability	absolute	3%

[2] In these papers [38, 39], the term "attribute" is considered as "concern".

The investigation by using frequency analysis technique also indicates that conflict between security and performance requirements are the most frequently conflicts discussed in the literature. 33.33% of the reviewed articles talk about this conflict, followed by conflict between security and usability requirements (23.33%) and conflict between availability and performance requirements (20%). This result indicates that those three types of conflicts (i.e. conflict between security and performance, between security and usability, and between availability and performance) are the three most frequent conflicts in the software projects and the most considered and essential to deal with in the software development process. The top ten conflicting NFRs that are often discussed in the literature are presented in Table 3.

5 Using the Catalogue

The catalogue of conflicts among NFRs, as presented in Fig. 3, extends and complements previously published NFRs conflicts models. Our work focuses on the extent and relativity of NFRs conflicts, that is, on negative links between NFRs and their corresponding-levels. Most of the existing conflicts models in the literature, however, concentrate on both positive and negative interrelationships. For example, Wiegers [9] has developed a matrix that represents the positive and negative relationships between particular type of NFRs; Egyed & Grünbacher [27] created a model of potential conflicts and cooperations among NFRs; and Sadana & Liu [37] have also defined conflict and support as the two types of contribution of a particular type of NFRs on the other types of NFRs.

Utilizing our NFRs catalogue of conflicts in conjunction with the existing conflicts models extends the overall understanding of how NFRs associate with each other (positive or negative) and how this negative association can be characterized in term of the relative characteristic of NFRs.

Software developers can use the conflicts catalogue to deal with various aspects of managing the conflicts among NFRs. For example, the conflicts catalogue can be used to identify which NFRs of the system that are really in conflict, including how relative the conflict is. If the identified conflict is an “absolute conflict”, then software developers may need to identify the potential strategies to resolve this conflict, such as prioritization strategy. On the other hand, if it is a “relative conflict”, then software developers need to understand and evaluate this particular NFRs in term of numerous factors involve in the development project (e.g. the meaning of particular type of NFRs in the context of the system being developed; the stakeholder involvement; or system development methodology used in the project) in order to further investigate whether those NFRs are really in conflict.

Furthermore, this catalogue can also be used to perform the NFRs conflicts analysis. By using this catalogue in conjunction with the framework presented by Sadana & Liu [37], software developers would be able to develop a structural hierarchy of functional and non-functional requirements affected by each conflict type. Therefore, this catalogue could further assist in the analysis of NFRs conflicts from the perspective of functional requirements. By utilizing this catalogue in conjunction with the “NFR Prioritizer” method presented by Mala & Uma [40], this catalogue could assist software developers to analyze the tradeoffs among NFRs and prioritize the NFRs. In term of analyzing the NFRs tradeoff, this catalogue can be used as the basis to develop the "NFR Taxonomy" that will be used to identify the type of relationships among NFRs. The NFR Taxonomy represents the conflicting or dependable association between each NFRs type. The example of NFR taxonomy is presented as follow [40]:

Usability#Accessibility+#Installability+#Operability+#Maintainability-

The above taxonomy represents that usability contributes positively to accessibility, installability, operability while it also contributes negatively to maintainability. Then, by combining the weight of user preference on each NFR type and the level of NFRs tradeoff derived from the NFR Taxonomy, software developers would be able to prioritize the NFRs of the system in term of the existence of conflicts among them.

Furthermore, this catalogue can also be used in conjunction with the "Trace Analyzer" technique developed by Egyed & Grünbacher [27]. The aim of this technique is to identify the true conflicts among NFRs of the system. By tracing the relationships between the system test cases and the software program codes, trace analyzer can characterize whether the conflicts listed in the NFRs conflicts catalogue are "really in conflict" in the developed system.

In term of conflicts resolution, the proposed catalogue of conflicts can also be used as the basis to execute a conflicts resolution technique. For example, by using this catalogue in conjunction with the "Non-Functional Decomposition (NFD)" framework developed by Poort & de With [41], software developers would be able to decompose the NFRs of the system when the NFRs conflicts identified.

6 Conclusions

Majority of techniques to manage the conflicts among NFRs present the documentation, catalogue, or list of potential conflicts. None of them deal with relative characteristic of NFRs. This relative characteristic means that the interpretation and importance of NFRs may vary depending on the particular system being developed as well as the extent of stakeholders' involvement. NFRs can be viewed, interpreted, and evaluated differently by different people and different contexts within which the system is being developed. Consequently, the positive or negative interrelationships among them are not always obvious.

In this paper we presented a catalogue of conflicts among NFRs by considering this relative characteristic. We presented the relativity of conflicts based on three categories: *absolute conflict; relative conflict;* and *never conflict.* This distinction would assist developers to perform further analysis of the identified conflicts and investigate the potential strategy to resolve the conflicts.

Furthermore, this catalogue can also be used to identify the NFRs conflicts in various phases of software development projects. For example, in the requirements engineering phase, during the elicitation process, system analysts would be able to identify which NFRs of the system will be in conflict and how relative this conflict is. This analysis would allow developers to identify the conflicts among NFRs early, so they would be able to discuss the potential conflicts with the system's stakeholder before specifying the software requirements. As another example, during the architecture design process, system designers could be able to use this catalogue to analyze the potential conflicts in term of the architectural decisions (e.g. layering, clustering, and modularity). The relativity of conflict relationships presented in the catalogue, would allow system designers to investigate the potential architecture strategies to get the best solution based on the type of conflicts among NFRs. Furthermore, by using this catalogue as the basis of conflicts identification, we can adopt numerous existing conflicts analysis and conflicts resolution techniques presented in the literature, such as [27, 37, 40, 41] to further investigate and evaluate the NFRs conflicts. Some examples of the existing techniques and the potential utilization of the catalogue in each technique have been described in Section 5 – "Using the Catalogue".

In the process of investigating conflicts and developing the conflicts catalogue, we also identified 114 NFRs types listed in the literature. Among these 114 types, more than 50% of the NFRs were introduced without any definition or attributes characterization while only 20% were provided with definition and attributes. This statistic and the list of NFRs types without definitions and attributes presented in this paper are expected to encourage software engineering community, particularly requirements engineering researchers to further investigate the unclear NFRs types and establish the a clear concept of them.

Further research will focus on collecting data from software practitioners to complete the catalogue. Those NFRs that have been removed from the initial catalogue due to lack of definitions and/or attributes will also be further investigated to improve the completeness of the catalogue. Also, the catalogue from industry can be compared with the one developed from the content analysis.

Moreover, besides collecting data to improve the conflicts catalogue, we would also perform further research on investigating the relative conflicts among NFRs. This study would not only investigate how those NFRs dynamically generate conflicts with each other in term of the system context, but also to develop a framework to assist developers in identifying in which situations those NFRs are in conflict and in which situations are not. The self-conflicting relationships will be covered in this study.

This study is conducted as part of a long term project of investigating conflicts among NFRs. Findings of this investigation, especially the conflicts catalogue, will be used as the basis to select those NFRs that are known to be frequently in conflict. The ultimate goal is to develop an integrated framework to effectively manage the conflicts between a pair of NFRs by considering the NFRs relative characteristic. This framework should be able not only to identify the existence and the extent of conflicts, but also to characterize and find the potential strategies to resolve the conflicts.

In this study, we do not claim that the catalogue of conflicts presented is an exhaustive and complete list. However, this catalogue represents what could be found in the current literature. We propose to conduct further research to compare and contrast our findings from the comprehensive review of research literature and the state of the practice.

Acknowledgements. This paper is an extended version of [42]. We would like to thank The International Schlumberger Foundation Paris for funding this research through Faculty for the Future Award Program.

References

1. Yeh, R.T.: Requirements analysis - a management perspective. In: IEEE Computer Software and Applications Conference (COMPSAC 1982), Los Alamitos, pp. 410–416 (1982)
2. Mairiza, D., et al.: Managing conflicts among non-functional requirements. In: 12th Australian Workshop on Requirements Engineering (AWRE 2009), Sydney, Australia (2009)
3. Ebert, C.: Putting requirement management into praxis: dealing with nonfunctional requirements. Information and Software Technology 40, 175–185 (1998)
4. Firesmith, D.: Using quality models to engineer quality requirements. Journal of Object Technology 2, 67–75 (2003)
5. Chung, L., et al.: Non-functional requirements in software engineering. Kluwer Academic Publishers, Massachusetts (2000)
6. Mittermeir, R.T., et al.: Modern software engineering, foundations and current perspectives. Van Nostrand Reinhold Co, New York (1989)
7. Kotonya, G., Sommerville, I.: Non-functional requirements (1998)
8. Charette, R.N.: Applications strategies for risk analysis. McGraw-Hill, New York (1990)
9. Wiegers, K.E.: Software requirements, 2nd edn. Microsoft Press, Washington (2003)
10. Sommerville, I.: Software Engineering, 7th edn. Pearson Education Limited, Essex (2004)
11. Breitman, K.K., et al.: The world's a stage: a survey on requirements engineering using a real-life case study. Journal of the Brazilian Computer Society 6, 1–57 (1999)
12. Finkelstein, A., Dowell, J.: A comedy of errors: the London ambulance service case study. In: Eigth International Workshop Software Specification and Design, pp. 2–5 (1996)
13. Boehm, B., In, H.: Identifying quality-requirements conflict. IEEE Software 13, 25–35 (1996)
14. Boehm, B., In, H.: Aids for identifying conflicts among quality requirements. IEEE Software (March 1996)
15. Leveson, N.G., Turner, C.S.: An investigation of the Therac-25 accidents. IEEE Computer 26, 18–41 (1993)
16. Grimshaw, D.J., Draper, G.W.: Non-functional requirements analysis: deficiencies in structured methods. Information and Software Technology 43, 629–634 (2001)
17. Heumesser, N., et al.: Essential and requisites for the management of evolution - requirements and incremental validation. Information Technology for European Advancement, ITEA-EMPRESS Consortium (2003)
18. Yusop, N., et al.: The impacts of non-functional requirements in web system projects. International Journal of Value Chain Management 2, 18–32 (2008)
19. Paech, B., Kerkow, D.: Non-functional requirements engineering - quality is essential. In: 10th International Workshop on Requirements Engineering: Foundation for Software Quality, pp. 27–40 (2004)
20. Lauesen, S.: Software requirements: styles and techniques. Addison-Wesley, Reading (2002)
21. Chung, L., et al.: Using non-functional requirements to systematically support change. In: The Second International Symposium on Requirements Engineering, York, pp. 132–139 (1995)
22. Chung, L., et al.: Dealing with change: an approach using non-functional requirements. Requirements Engineering 1, 238–260 (1996)
23. Curtis, B., et al.: A field study of the software design process for large systems. Communication of the ACM 31, 1268–1287 (1988)

24. Boehm, B., Egyed, A.: WinWin requirements negotiation processes: a multi-project analysis. In: 5th International Conference on Software Processes (1998)
25. Egyed, A., Boehm, B.: A comparison study in software requirements negotiation. In: 8th Annual International Symposium on Systems Engineering, INCOSE 1998 (1998)
26. Robinson, W.N., et al.: Requirements interaction management. ACM Computing Surveys 35, 132–190 (2003)
27. Egyed, A., Grünbacher, P.: Identifying requirements conflicts and cooperation: how quality attributes and automated traceability can help. IEEE Software 21, 50–58 (2004)
28. Krippendorff, K.: Content analysis: and introduction to its methodology, 2nd edn. Sage Publications, Inc., Thousand Oaks (2004)
29. Weber, R.P.: Basic content analysis. Sage Publications, Inc., Thousand Oaks (1989)
30. Stemler, S.: An overview of content analysis. Practical Assessment, Research & Evaluation 7 (2001)
31. Neuendorf, K.A.: The content analysis guidebook, 1st edn. Sage Publications, Inc., Thousand Oaks (2001)
32. Glinz, M.: Rethinking the notion of non-functional requirements. In: Third World Congress for Software Quality, Munich, Germany, pp. 55–64 (2005)
33. Glinz, M.: On non-functional requirements. In: 15th IEEE International Requirements Engineering Conference (RE 2007), pp. 21–26 (2007)
34. Mairiza, D., et al.: An investigation into the notion of non-functional requirements. In: 25th ACM Symposium On Applied Computing, Switzerland (2010)
35. Alexander, I., Maiden, N.: Scenarios, stories, use cases: through the systems development life-cycle. John Wiley & Sons, Ltd., Chichester (2004)
36. Robertson, S., Robertson, J.: Mastering the requirements process, 2nd edn. Addison-Wesley, Boston (2006)
37. Sadana, V., Liu, X.F.: 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. Brito, I., Moreira, A.: Integrating the NFR framework in a RE model. Presented at the Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, Lancaster, UK (2004)
39. Moreira, A., et al.: Crosscutting quality attributes for requirements engineering. In: 14th International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy (2002)
40. Mala, G.S.A., Uma, G.V.: Elicitation of non-functional requirements preference for actors of usecase from domain model. In: Hoffmann, A., Kang, B.-h., Richards, D., Tsumoto, S. (eds.) PKAW 2006. LNCS (LNAI), vol. 4303, pp. 238–243. Springer, Heidelberg (2006)
41. Poort, E.R., de With, P.H.N.: Resolving requirement conflicts through non-functional decomposition. In: Fourth Working IEEE/IFIP Conference on Software Architecture, WICSA 2004 (2004)
42. Mairiza, D., et al.: Towards a catalogue of conflicts among non-functional requirements. In: Maciaszek, L.A., Loucopoulos, P. (eds.) ENASE 2010. CCIS, vol. 230, pp. 33–46. Springer, Heidelberg (2011)