



# Non-Functional Requirements Elicitation and Incorporation into Class Diagrams

Xiaoyu Song, Zhenhua Duan, Cong Tian

## ► To cite this version:

Xiaoyu Song, Zhenhua Duan, Cong Tian. Non-Functional Requirements Elicitation and Incorporation into Class Diagrams. 6th IFIP TC 12 International Conference on Intelligent Information Processing (IIP), Oct 2010, Manchester, United Kingdom. pp.72-81, 10.1007/978-3-642-16327-2\_12 . hal-01055075

**HAL Id: hal-01055075**

**<https://inria.hal.science/hal-01055075v1>**

Submitted on 11 Aug 2014

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

# Non-Functional Requirements Elicitation and Incorporation into Class Diagrams<sup>\*</sup>

Xiaoyu Song, Zhenhua Duan and Cong Tian

Institute of Computing Theory and Technology, and ISN Laboratory, Xidian University, Xi'an,  
710071, P.R. China

Email: yming.song@gmail.com, {zhhd, ctian}@mail.xidian.edu.cn

**Abstract.** Top-quality software architecture should consider both functional and non-functional aspects of systems and their association. In the existing literature, considerable efforts have been directed at functional requirement analysis and design, regardless of the non-functional aspects. This disassociation makes architecture comprehension and evolution hard. This paper proposes a strategy on how to elicit non-functional requirements and incorporate them into the design models of functions. We aim at bridging the gap between functionality and non-functionality and constructing high quality software systems.

## 1 Introduction

The software development is concerned with two different aspects of a system: functional and non-functional ones. The development of functions has made great progress, and it has been supported by many development approaches. In contrast, there are only a few researchers working on non-functions, and even no tools completely support the non-functional development in software engineering. However, non-functional requirements (NFRs) have an important effect on system quality and development costs. The well-known case of the London Ambulance System (LAS) is a good example [1]. The LAS was deactivated, because of several problems related to non-functional requirements. The market is increasing its demands on software that not only implements all of functionalities but also copes with non-functional aspects such as reliability, security, accuracy, safety, performance as well as others [2].

NFRs, in spite of importance, are usually hidden in developers mind. However, it does not mean that software engineers cannot consider information about non-functional requirements [1]. Moreover, NFRs are always linked to functional requirements (FRs). We raise a systematic methodology for software development. It contains the following main processes. First, we capture all requirements from users and customers, and form requirement documents; secondly, we analyze requirement documents and elicit both FRs and NFRs; thirdly, we construct function models based on FRs; fourthly, we incorporate the elicited NFRs into function models. This results in the association of system's

---

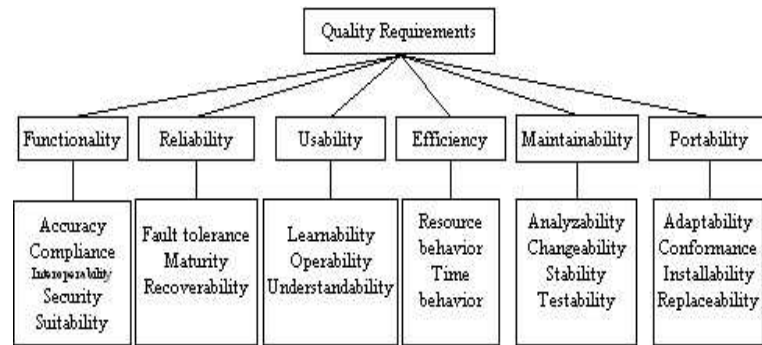
<sup>\*</sup> This research is supported by the NSFC Grant No. 60373103, 60433010, 60873018 and 60910004, DPRPC Grant No. 51315050105, 973 Program Grant No. 2010CB328102 and SRFDP Grant No. 200807010012.

functionality and non-functionality, improving the quality of software and the speed of development.

In this paper, we adopt an object-oriented approach based on UML. It gives an expression to system's functions. The *use case model* of UML constitutes a suitable description of system's functionality [3] while *class diagrams* of UML express the design model of system's functions.

Our work focus on eliciting NFRs and incorporating NFRs into the design models of system's functions. To elicit NFRs, we first extract the words or phrases regarding non-functional aspects in requirement documents; then we define and enrich the resulted NFRs. The incorporation process is to incorporate NFRs into class diagrams.

The remainder of this paper is constructed as follows: Section 2 introduces the related basic concepts. Section 3 depicts the strategy and the process of the elicitation and integration. Section 4 gives an example. Finally, the conclusion and future work are discussed.



**Fig. 1.** ISO/IEC 9126 Taxonomy of Quality Requirement

## 2 Basic concepts

### 2.1 Non-functional requirements

Software requirements consist of two different requirements, the functional requirements and the non-functional requirements. FRs focus on "what" the system must do while NFRs constrain "how" the system can accomplish the "what". So it is said that NFRs are always related to FRs in software development.

NFRs are also known as Quality Requirements [4]. A set of ISO/IEC standards are related to software quality, being standards number 9126 (which is in process of substitution by 9126-1, 9126-2 and 9126-3), 14598-1 and 14598-4 the more relevant ones [5]. The ISO/IEC 9126 standard prescribes the top characteristics: functionality, reliability, usability, efficiency, maintainability and portability. And the top characteristics can be further refined into subcharacteristics (see figure 1). The top characteristics of

NFRs have a high level of abstraction while the subcharacteristics have a greater level of detailed aspects.

Each NFR should belong to a NFR type. In this work, it is decided that the set of subcharacteristics of NFRs instead of the top characteristics is as the foundation of categorizing NFRs.

## **2.2 NFR Card**

An NFR card is used to record information on an NFR, metaphorized into the belt to connect an NFR and class diagrams.

An NFR card is composed of four parts, NFR symbol, NFR property, NFR behavior and Incorporated Class (see figure 4 and figure 5). An NFR symbol is the name or alias of the meaningful word or phrase referring to NFR, while an NFR type expresses the category an NFR belongs to. The property an NFR possesses is called an NFR property and the behavior satisfying an NFR is called NFR behavior. Incorporated class is what NFRs can be incorporated into. The NFR property plays a role as the class attribute does in a class. Each NFR should contain at least one NFR behavior which implements the NFR. There may be no incorporated Class, because each class in class diagrams may be not related to the NFR. However, it is not to say that the incorporation is failure. In this case, a new class on NFR can be inserted into class diagrams. The detail will be described in Section 3.2.

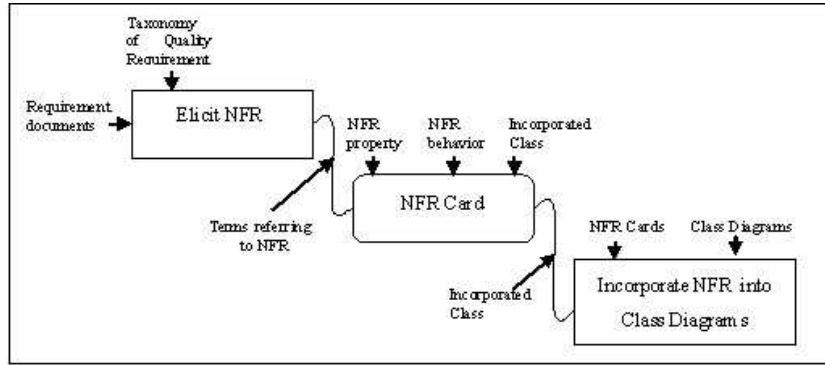
## **3 The proposed strategy**

In this section, we propose a strategy to deal with NFRs from eliciting to incorporating into class diagrams. NFRs can be elicited based on requirement documents, because these documents may contain the words or phrases referring to NFRs. Then an NFR card will be constructed as a belt to connect the elicitation process and the incorporation process. In an NFR card, it is indicated that what properties and behaviors are responsible for satisfying the NFR, and what class in class diagrams is related to the NFR. Once finding out the incorporated class, we insert the NFR into the class. But there may be an exception that the incorporated class does not exist. In this case, we will add a new class which satisfies the NFR to class diagrams. Figure 2 shows the structure for our strategy.

### **3.1 Eliciting non-functional requirements**

Our approach is useful for eliciting non-functional requirements. NFRs are not as clear in stakeholders' minds as functional requirements, so eliciting NFRs calls for the apprehension of the domain and the accumulation of knowledge and skills.

The good approach to elicit NFRs is to read those accomplished requirement documents carefully and identify NFRs with the accumulated experience. These documents record all requirements desired by users and customers and usually contain some words or phrases related to NFRs. The first step is to extract these words or phrases from



**Fig. 2.** the structure for our strategy

requirement documents. Then, we refine them into the NFR symbols. One NFR symbol should express an object which possesses non-functional characteristics. The NFR symbols are just the possible ones and they need to be validated by users and customers later on.

Each NFR should be placed in an NFR type. As mentioned in Section 2.1, one NFR type is one sub-characteristic of Quality requirements. We categorize all possible NFR symbols to different NFR types, such as accuracy, security and testability.

All NFR properties and NFR behaviors for a possible NFR symbol may not be perceived in the first time. They can be completely captured by interviewing the related stakeholders or sending them questionnaires.

It is very necessary to validate these already elicited NFRs, and their properties and behaviors, because accuracy of these NFRs has a vital impact on the software architecture. It needs the attendance of the related stakeholders to confirm which NFRs elicited are the ones desired by users and customers. Moreover, the correctness and maturity of NFR properties and NFR behaviors should be validated any more.

In order to obtain authentic NFRs, those activities, such as extracting NFRs, capturing NFR properties and behaviors and validating NFRs, should be carried out several times.

The next step is to build the NFR card for each NFR. An NFR card describes the information on an elicited NFR. The NFR symbol is used to name the NFR and a sub-characteristic of quality requirements is marked as the NFR type. In addition, NFR properties and NFR behaviors should be organized and formalized in the NFR card. Those collected NFR properties and behaviors from users and customers may be informal at first. In order to facilitate the incorporation, they have to be formalized.

The format of NFR property is

*PropertyName : propertyType, set value = propertyValue*

The format of NFR behavior is

*BehaviorName : Parameter1, ..., ParameterN*

For instance, there is an informal sentence describing NFR property,

*"Response time of the transaction must be not longer than 5s".*

In accordance with the format above, we can acquire

*ResponseTime* : *s*, *set value* = 5

### 3.2 Incorporating NFRs into Class Diagrams

Incorporating Non-functional requirements into class diagrams is the significant contribution of our work. Eliciting NFRs is important but is not enough [1]. We have to incorporate NFRs into the design models of system's functions. As class diagrams give the primary functional representation for a system, the incorporation of NFRs into them will make the system architecture more systematic.

An NFR card plays an important role for the proposed strategy, as output of the elicitation and input of the incorporation. In the elicitation process, the NFR symbol, properties and behaviors have been accomplished. Hence, searching for the incorporated class in class diagrams is the constant activity. Since system non-functionality needs to refer to system functionality and class diagrams are usually used to model system's function, so there may be a relation between class diagrams and NFRs. This is to say, class diagrams may contain the incorporated class that NFRs can be incorporated into. However, it is unassailable to find out the incorporated class for every NFR. The incorporated class for some NFR is not present.

We discuss the incorporation process in two cases: the one case is that the incorporated class exists and the other case is that the incorporated class does not exist.

Suppose that the incorporated class exists and it has been appended to the NFR Card. Based on the incorporated class, the NFR properties and behaviors will be inserted into the class as new attributes and new operations. In order to distinguish the new attributes and operations from the old ones in a class, we attach the postfix {NFR-Type [NFR-Name]} to the new attributes and operations, for instance, Generate-Alarm () {Security [Credit Card]}.

We propose to use the following processes to incorporate NFRs into class diagrams.

1. Pick up an NFR card and search for the class that the NFR can be incorporated into. Since the NFR symbol is identified from requirement documents and class diagrams are also constructed based on these documents, it is possible that some class contains the NFR symbol. We consider the NFR symbol as the starting point to search for the incorporated class, because the symbol from requirement documents may be related to both the functional aspects and the non-functional aspects. Of course, the NFR properties and behaviors may be also helpful. When found out, the class would be marked as the incorporated class in an NFR card.

2. Incorporate the NFR into the incorporated class. Each NFR property is translated into a new attribute while each NFR behavior is translated into a new operation. The NFR symbol combines with NFR type to form the postfix {NFR-Type[NFR-symbol]}. Both NFR property and NFR behavior must be formalized because it is convenient for the automation of the incorporation (see figure 3).

Some NFRs are not directly related to any class. We call those requirements global non-functional requirements (GNFRs). Incorporating GNFRs into class diagrams is also absolutely necessary for software development. In this case, every GNFR will be translated into a new class with NFR symbol as class name. Furthermore, the NFR prop-

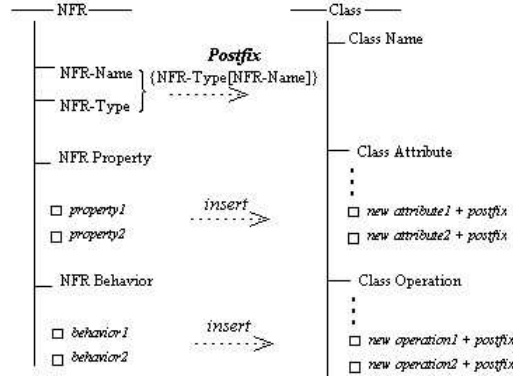


Fig. 3. incorporate NFR into class diagram

erties and behaviors are also translated into class attributes and operations as mentioned above.

After incorporating NFRs into class diagrams, we propose to adjust class diagrams, because the addition of NFRs can lead to some changes of the structure of class diagrams. However, for the limitation of space, we do not discuss it in detail in the paper.

#### 4 An example for the proposed strategy

The example used to validate the proposed strategy is about the development of a credit card system. Here is a segment of requirement documents for this system. "In the Credit Card system, it is necessary to protect against the vicious access to the system. Security of the transaction must be satisfied. ... Moreover, the response time of the transaction is not longer than 5s".

We start to analyze the above sentence to identify the non-functional requirements for this system. After analyzing thoroughly, we find the phrases "security of the transaction" and "response time of the transaction" are related with NFRs. So we obtain two NFRs: i) security of the transaction; ii) response time of the transaction.

As each NFR has an NFR card, we have to construct two NFR cards. In the example, it is accidental that both of the NFRs are based on the same object *transaction*, so the word *transaction* is considered as NFR symbol for the two NFRs. *Security* and *Time behavior* are two NFRs Types.

The next step is to confirm NFR properties and NFR behaviors. This step depends on not only software engineers but also the other stakeholders through interviewing them or sending them questionnaires. For the NFR with the type *Security*, we discover that "alarm" is used to notify the proper authority of all vicious accesses to the credit card. So there is one NFR behavior "send alarm" to implement the NFR. For the NFR with the type *Time behavior*, the property *Response Time* and the behavior *halt the transaction* are asked to satisfy the requirement about response time of the system.

Validating NFR properties and NFR behaviors is necessary, and it requires all related stakeholders to work together. In our work, the result of NFRs validation demonstrates that the identified NFR property and behaviors are correct.

**Fig. 4.** the NFR Card with type Security

Accomplishing the NFR Cards is the next activity. It has to be done to formalize the NFR properties and behaviors as introduced in Section 3.1. Figure 4 and 5 portray two NFR cards.

The elicited NFRs will be incorporated into class diagrams. Figure 6 shows a partial class diagram for the Credit Card system. There is a class whose name is the same as the NFR symbol in this diagram and there is a close relation between the class and two NFRs, so we consider class *Transaction* as the incorporated class. until now, the two NFR Cards have been constructed successfully.

For the NFR with *Security* type, the behavior *SendAlarm* is translated into a class operation *SendAlarm ()*. And the postfix {Security [Transaction]} will be attached to the operation. For the NFR with *Time behavior* type, the property *Response Time* is translated into a class attribute *ResponseTime* while the behavior *halt the transaction* is translated into a class operation *HaltTransaction()*. Their postfixes are {Security [Time behavior]}; Figure 7 shows the class *Transaction* after the NFR integration.

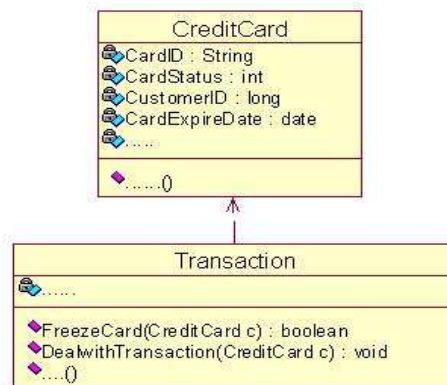
## 5 Related works

In spite of its importance, NFRs have surprisingly received little attention in the literature, and they are poorly understood in contrast with other less critical aspects of the software development [6]. Recently, some researches concerning NFRs have made progress. There are two main approaches in the software development concerning with NFRs. On one hand, the majority of the studies on NFRs use a product-oriented approach, which is mainly concerned with how much a software is in accordance with

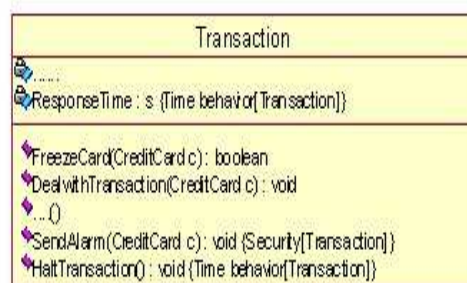




**Fig. 5.** the NFR Card with type Time behavior



**Fig. 6.** a partial class diagram of the Credit Card system



**Fig. 7.** class Transaction after NFR integration

the set of NFRs that it should satisfy [7] [8] [9] [10]. On the other hand, there are also a little studies proposing to use a process-oriented approach in order to explicitly deal with NFRs. Unlike the product-oriented approach, this approach is concerned with making NFRs a relevant and important part of the software development process. Chung's Framework [6] is one of complete works. In his work, NFRs are viewed as goals that are decomposed into subgoals until one finds that all the necessary actions and information are already represented at the leaf levels of the graphs. Although Chung's work aims to represent NFRs and their conflicts, it does not consider the association with functional requirements. In our work, we believe that the systematic elicitation and integration of NFRs into design models may be helpful in software development.

## 6 Conclusion

Errors resulting from NFRs are the most expensive and difficult to correct [11], and not dealing or improperly dealing with NFRs can lead to more expensive software and a longer time-to-the market [12]. So it is strongly demanded to satisfy all NFRs for a system and deal with both FRs and NFRs and their association through the entire development process.

Our current work fills the gap between system functionality and system non-functionality in software development, and also applies NFRs to one design model of system's functions. We raise a systematic methodology for software development. The paper mainly focus on the strategy to elicit NFRs and incorporate NFRs into class diagrams. The result shows that integrating NFR into the design of system's functions can consummate system architecture, improve system quality and reduce failure risk.

However, we only consider the incorporation between NFRs and class diagrams. In the future, we will extend the strategy of the incorporation to the other design models, such as sequence diagrams and state machine diagram. We have developed a prototype to support the proposed strategy. However, it has not achieved automation of the incorporation process and still requires software engineer's attendance. Further, we will improve the tool so that our strategy can be better supported.

## References

1. LuiZ Marcio Cysneiros, Julio Cesar Sampaio do Prado Leite and Jaime de Melo Sabat Neto. A Framework for Integrating Non-Functional Requirements into Conceptual Models. *Requirements Eng* (2001) 6:97-115 @ 2001 Springer-Verlag London Limited.
2. G.S. Anandha Mala and G.V. Uma. Requirement Preference for Actors of Usecase from Domain ModelPKAW 2006, LNAI 4303, pp. 238 - 243, 2006.
3. Haeng-Kon Kim and Youn-Ky Chung. Automatic Translation from Requirements Model into Use Cases Modeling on UML. O. Gervasi et al. (Eds.): ICCSA 2005, LNCS 3482, pp. 769 - 777, 2005. @ Springer-Verlag Berlin Heidelberg 2005.
4. Chung L. Representing and Using Non-Functional Requirements: A Process Oriented Approach. Ph.D. Thesis, Dept. of Comp.. Science. University of Toronto, June 1993. Also tech. Rep. DKBS-TR-91-1.
5. ISO/IEC Standards 9126 (Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for their use, 1991) and 14598 (Information Technology - Software Product Evaluation: Part 1, General Overview; Part 4, Process for Acquirers; 1999).

6. Chung, L., Nixon, B., Yu, E. and Mylopoulos, J. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers 2000.
7. Fenton, N.E. and Pfleeger, S.L. Software Metrics. A Rigorous and Practical Approach. International Thomson Computer Press, 1997.
8. Kirner T.G., Davis A .M., Nonfunctional Requirements of Real-Time Systems, Advances in Computers, Vol 42 pp 1-37 1996.
9. Lyu, M.R. (ed.) Handbook of Software Reliability Engineering. McGraw-Hill, 1996.
10. Musa, J., Lannino, A. and Okumoto, K. Software Reliability: Measurement, Prediction, Application. New York, McGraw-Hill, 1987
11. Brooks Jr., F.P. No Silver Bullet. Essences and Accidents of Software Engineering. IEEE Computer Apr 1987, No 4 pp: 10-19, 1987.
12. Luiz Marcio Cysneiros, Julio Cesar Sampaio do Prado Leite. Using UML to Reflect Non-Functional Requirements. Proc. of the CASCON '01, Toronto, 2001-11.