A Survey of Non-Functional Requirements in Software Development Process

Abderrahman Matoussi, Régine Laleau

A Survey of Non-Functional Requirements in Software Development Process

Abderrahman Matoussi **Régine Laleau**

October 2008

TR–LACL–2008–7

Laboratoire d'Algorithmique, Complexité et Logique (LACL)
Département d'Informatique
Université Paris 12 – Val de Marne, Faculté des Science et Technologie
61, Avenue du Général de Gaulle, 94010 Créteil cedex, France
Tel.: (33)(1) 45 17 16 47, Fax: (33)(1) 45 17 66 01

Laboratory of Algorithmics, Complexity and Logic (LACL)
University Paris 12 (Paris East)

Technical Report **TR–LACL–2008–7**

A. Matoussi, R. Laleau.

A Survey of Non-Functional Requirements in Software Development Process

A Survey of Non-Functional Requirements in Software Development Process

Abderrahman Matoussi **Régine Laleau**

Laboratory of Algorithmics, Complexity and Logic - University Paris 12 (Paris East), France
{abderrahman.matoussi,laleau}@univ-paris12.fr

Abstract

Due to the enormous pressure towards deploying software as fast as possible, functional requirements have been the main focus of software development process at the expense of implementing non-functional requirements (NFRs) such as performance and security. Thus, in practice, NFRs have been observed to be frequently neglected or forgotten in the software development process. However, NFRs is an important concept in requirements engineering which plays an essential role in the success or the failure of systems. NFRs introduce quality characteristics, but they also represent constraints under which the system must operate. So, the chances of success for the software system are maximized when NFRs are modeled since the initial phases of the development process. This article reviews the NFR concepts, relates them to the overall software development process and identifies new areas of further work.

Keywords. Requirements engineering, Non-functional requirements, Software development process.

1 Introduction

It is clear that the primary measure of success of a software system is the degree to which it meets the purpose for which it was intended. In this sense, requirements engineering is the process of discovering that purpose by identifying stakeholders and their needs. Several definitions of requirements engineering have been proposed in the literature. The most used one is the one proposed by [Zav97] "Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families". It is judicious to affirm that it is an attractive definition for a number of reasons, related especially to the notion of *precise specifications*. These last ones provide the basis for analysing requirements, validating them, defining what designers have to build and verifying that they have done so correctly upon delivery. Thus, requirements specification is a critical task because if we introduce mistakes , it is difficult and expensive to remove them afterwards. For that, it is important to be interested in the way these requirements are specified. There are two extreme ways of specifying software requirements: a complete formal specification and an informal specification. There also exist some intermediate representations that have shown to be useful in communication between users and specialists. In spite of it offers better results, formal specification has been often neglected in the literature because it is still perceived as more difficult and expensive (time and expert person). We think that is crucial to define the notion of formal specification in order to emphasis its importance into the requirements engineering. Hence, we note that formal specification is the expression, in some formal language and at some level of abstraction, of a collection of properties some system should satisfy. Moreover, we can note that the act of formalization in itself has been widely experienced to raise many questions and detect serious problems in original informal formulations. According to [Lam00], a specification is formal if it is expressed in a language made of three components: (i) rules for determining the grammatical of sentences (the syntax); (ii) rules for interpreting sentences in a precise (the semantics); (iii) rules for inferring useful information from the specification (the proof theory). It is also important to indicate that there are different levels in the software lifecycle at which formal specification may enter the picture: when modelling the domain, when designing a functional model for the software, when designing the software architecture or when modifying or re-engineering the software.

Non-functional requirements is another important concept in the requirements engineering which plays an essential role in the success or the failure of systems. The NFRs concept has been widely studied by the community. Several definitions of NFR exists in the literature. IEEE defines Non-Functional Requirements as "a software requirement that describes not what the software will do, but how the software will do it, for example, software performance requirements, software external interface requirements, design constraints, and software quality attributes". In order to better understand the concept of NFR, [Gli07] constitutes an excellent reference which surveys all the definitions of the NFR term in the last 20 years and discusses all the limits of these definitions. The variety of the researches which deal with NFRs allow us to affirm that they are now part of several requirements engineering syllabus. Up to now, the majority of these researches have not focused on NFRs as first class requirements. In fact, incorporating NFRs into the different phases of the software life-cycle is a very hard task. Researchers face many challenges including great diversity of NFRs, NFRs subjective nature, formally specifying requirements, incorporating these requirements into models used for specifying functional requirements and resolving conflicts among NFRs. The latter is a very important challenge. Indeed, requirements do not manifest in isolation and normally satisfying one NFR may affect negatively the satisfaction of another. For instance, when we decide to use a double password to ensure security (one NFR), it can impact the system performance (another NFR) because checking two different passwords slow down the system. Despite this obvious importance and relevance of NFRs, NFRs are almost always left to be verified after the implementation is finished, which means NFRs are not mapped directly and explicitly from requirements engineering to implemetnation. This leaves software development with potential exacerbation of the age-old problem of requirements errors that are not detected until very late in the process. The authors of [NLC00] enumerate some of the well-known problems of the software development due of the NFRs omission: cost and schedule overruns, software systems discontinuation and dissatisfaction of software systems users. For all that, it is important to affirm that NFR should affect all levels of software life cycle and shall be identified as soon as possible and their elicitation must be accurate and complete. In the rest of the paper, we are going to review the background information on especially existing NFR related work at the requirements analysis and specification level, the design level and the implementation level.

The remainder of the paper is organized as follows. Section 2 reviews the state of the art that handles NFRs in the areas of requirement analysis and specification. Section 3 focuses on the design level. Section 4 concentrate on the implementation level. Section 5 highlights the complete approaches that consider NFRs through the whole process of software development. Finally, Section 6 draws some conclusions and highlights current limits.

2 Requirements analysis and specification level

It is clear that a current industrial practice is to specify only functional requirements at the first levels of software development and the NFRs at the implementation level. Consequently, initial levels (analysis and specification) of a software system may not reflect the NFRs properly and may lead to a failed product. In order to resolve this problem, international standard are recommending the consideration of NFRs during its development process, beginning from the step in which requirements are specified in detail. Different approaches have been proposed ranging from unstructured and informal text to highly formal mathematical approaches. The approach used in each case depends on the goals of the project and the available resources.

2.1 Informal and semi-formal approaches

The main interest of informal and semi-formal approaches is that they don't need a high personal expertise. Thus, it is easy to use these approaches. However, these approaches provide very little support for system analysis; i.e. there is no certainty that the obtained specifications are complete or no ambiguous. A survey of the literature found that most people use informal or semi-formal approaches to specify NFRs because the formal ones are still perceived as more difficult and expensive. In what follows, we present some informal and semi-formal methods presented in the literature to analyze and specify both NFRs and functional requirements.

The NFR Framework [CNYM00] is the most popular work in this topic. It treats NFRs as soft goals (goals that are hard to express) to be addressed during the development process. NFRs, design decisions and their relations are captured in a goal graph where the nodes are either NFRs or design decisions. Goals in NFR Framework can be refined into detailed concrete goals. NFR Framework makes the relationships between NFRs and intended decisions explicit. This helps better understand the impact of every design decision; i.e. typically one design decision may impact multiple NFRs positively or negatively. The main interest of this framework is that it can reused by other models to handle NFRs. For instance, [KDO07] presents a systematic and precisely semi-formal model that extends the taxonomy of the NFR Framework by integrating the concept of "hard goals NFRs" within the requirements engineering process. This model details the sequence of systematic activities towards an early consideration of identifying, specifying and separating FRs and NFRs: (i) requirements elicitation that aims to discover the requirements for the system; (ii) analysis and crosscutting realization; (iii) composing requirements that integrate identified crosscuttings (both functional and non-functional) with the use-case model and the domain model.

In [KK99], the authors present a new informal procedure for requirements engineers to elicit requirements specification with its evaluation of stakeholders. Since many kinds of stakeholders participate in one system development, some stakeholder requirements conflict with others. To coordinate the requirements among stakeholders or to find a trade-off, criteria shared by the stakeholders are required. [KK99] affirm that NFRs can play this role because they influence the concerns of most of stakeholders. Hence, NFRs are used for checking stakeholders satisfaction. With this procedure, stakeholder-dissatisfaction can be reduced and new possibilities to satisfy or dissatisfy other stakeholders can be found. The outputs of this procedure are the refined specifications written in a sequence diagram and evaluation tables which are used to store the evaluations of each stakeholder. The rows in an evaluation table correspond to requirements types. The columns of the table correspond to the kinds of stakeholders. Each cell in the table shows a stakeholders evaluation by the type of requirement. Each cell is labeled by three attributes: a reference of a refined specification, a score and the content of the evaluation. We think that the notation of this procedure is simple but useful for stakeholders to validate the correctness of specifications on the spot. However, this procedure is not suitable for the cases where requirements engineers can not interactively elicit requirements from stakeholders. We think also that priority or bias should be given to some kind of NFRs, or to some kind of stakeholders, when a specific analysis is needed.

The quality attributes taxonomy [BKLW95] is another semi-formal work which is the result of CMU SEI's research work on how quality attributes impact a software architecture. The interest of this work is that the terminology used in the taxonomy can serve as a vocabulary to specify NFRs and then drive the design of the architecture. The taxonomy is divided into four areas: performance, dependability, security, and safety. All quality attributes are analyzed through three dimensions: concerns, factors, and methods. Firstly, concerns or requirements are the parameters by which the attributes of a system are specified. Secondly, factors are the properties of the system and its environment that have an impact on the concerns. Factors might not be independent and might have cause/effect relationships. Thirdly, methods specify how to address the concerns.

The quality attributes taxonomy has inspired many other works. For instance, [DKKP03] presents a semi-formal approach for eliciting and documenting efficiency requirements (a type of NFRs) with use cases and a high-level architecture. This approach uses quality attributes and also quality models to capture general knowledge on NFRs, while specific NFRs are captured in a template. The interest of this approach is that it uses a generalized meta-model and a quality model that can then be used for other NFRs.

Agile development methods are often criticized for not having explicit practices for eliciting NFRs [PEM03]. However, [HJWM06] discuss an agile approach to address the specification and testing of performance which is an important type of NFRs. Thus, [HJWM06] proposes an evolutionary model called software performance requirements evolution model PREM). Using the model as a guideline, a development team can identify and specify performance requirements incrementally, starting with casual descriptions, and refine them to a desired level of detail and precision.

Misuse cases [HP05, SO05, Fir03] are other approaches, based on UML, to handle NFRs such as security requirements. Misuse cases are the inverse of use cases and describe functions that the system should not allow. In [HP05], the authors outline concepts for an analysis of NFRs of a software system and develop a misuse based method for deriving detailed NFRs but also functional requirements. The authors believe that the analysis of NFRs in terms of assets, threats, misusers and countermeasures helps to complement software and project requirements. The approach presented in [Fir03] differentiate between misuse cases that are more appropriate for analyzing and specifying security threats and security use cases that should be used to analyze and specify security requirements (see Figure 1). Misuse cases and security use cases inherit the simplicity and popularity as well as the semantic inconsistencies of UML.

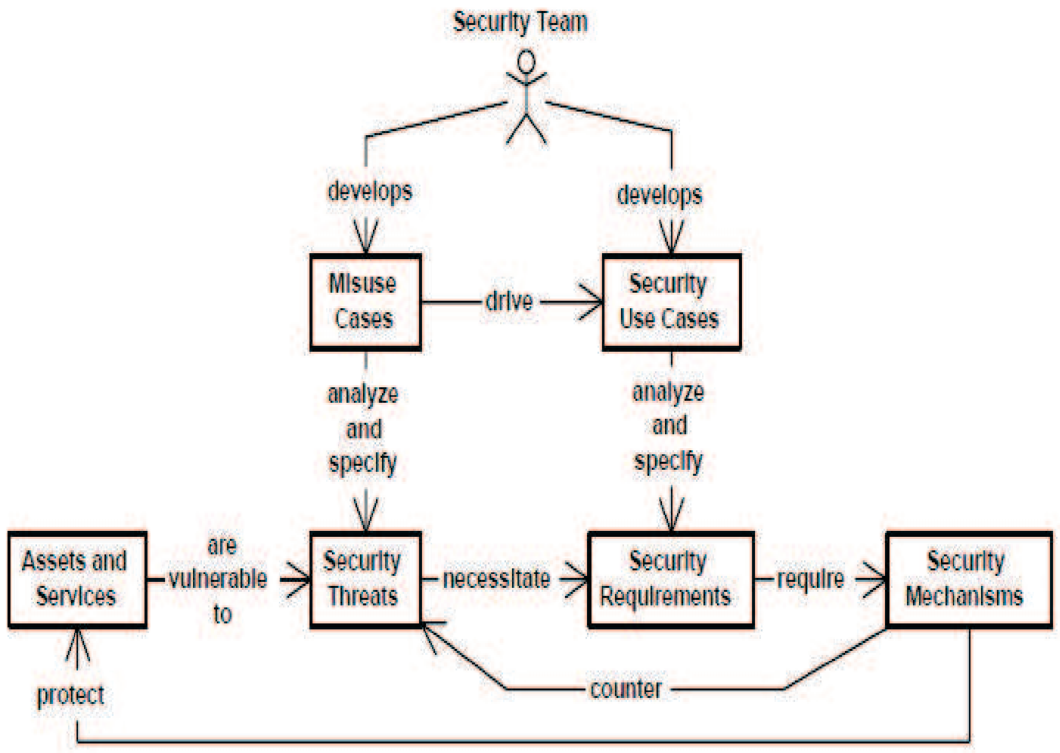

Figure 1: Misuse Cases and Security Use Cases [Fir03]

KAOS(Knowledge Acquisition in autOmated Specification) [Lam01, Let01] is another semi-formal approach for eliciting, analyzing and modeling essentially functional requirements. KAOS requirements model is based on first-order temporal logic and it is composed of several sub-models related through inter-model consistency rules. [Lam04] has then extended KAOS to handle NFRs such as security requirements. Thus, the author specifies some generic specification patterns for eliciting security goals such as confidentiality, integrity, availability, privacy, authentication and non-repudiation. The general method for elaborating KAOS security requirements [Lam04] is based on the incremental building and specification of two concurrent models: an intentional model of the system-to-be and an intentional anti-model yielding vulnerabilities and capabilities required for achieving the anti-goals of threatening security goals from the original model. The method allows threat trees to be derived systematically through anti-goal refinement until leaf nodes are reached that are either attack vulnerabilities observable by the attacker or anti-requirements implementable by this attacker. The original model is then enriched with new security requirements derived as countermeasures to the anti-model.

2.2 Formal approaches

Formal NFRs elaboration method written in a formal language became more and more a necessity. However, a familiar problem with formal methods in specifying such requirements is the high cost and difficulty of using them. Researchers justify this cost by applying formal methods only to crucial NFRs such as security requirements. Indeed, the most NFRs are critical aspects of the system that may otherwise result in huge losses in terms of time money and data when poorly specified. In the following, we present some research works on the formalization of NFRs.

Formal Design Analysis Framework (FDAF) [CDDD03] is intended to support the systematic design of a system that meets its NFRs such as performance, security, etc. A variety of notations have already been used to describe architectures including UML (standard, formalized, with OCL, and extended) and formal methods to support the rigorous analysis of a design. An overview of the FDAF is illustrated in Figure 2. In this figure, the FDAF is represented with a cloud surrounded by the stakeholders, inputs, and outputs of the framework. The stakeholders are the designers, requirements engineers, and formal methodologists who use the framework to develop and evaluate a system design that meets its functional and non-functional requirements. The inputs include an object-oriented design model documented in the UML notation and a requirements specification that includes the NFRs and FRs for the system. FDAF assist the user in selecting a formal method, and translates an extended semi-formal UML design into a formal notation. The major limit of FDAF is that the variety of formal notations used to formalize NFRs constitute an obstacle to study the different interactions between NFRs. Consequently, the user cannot detect conflicts between the different types of NFRs.

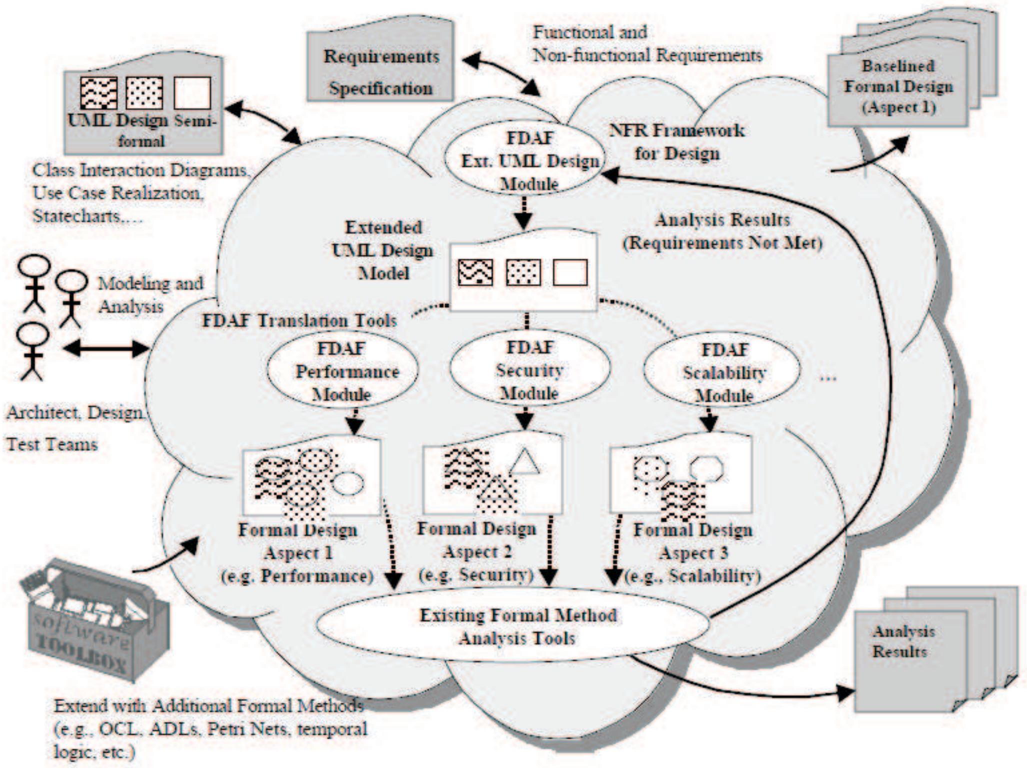
Requirements Specification
Functional and Non-functional Requirements
Baselined Formal Design (Aspect 1)
UML Design Semi-formal
Class Interaction Diagrams, Use Case Realization, Statecharts,...
FDAF Ext. UML Design Module
NFR Framework for Design
Analysis Results (Requirements Not Met)
Extended UML Design Model
Modeling and Analysis
FDAF Translation Tools
FDAF Performance Module
FDAF Security Module
FDAF Scalability Module
...
Architect, Design, Test Teams
Formal Design Aspect 1 (e.g. Performance)
Formal Design Aspect 2 (e.g. Security)
Formal Design Aspect 3 (e.g., Scalability)
Existing Formal Method Analysis Tools
software TOOLBOX
Analysis Results
Extend with Additional Formal Methods (e.g., OCL, ADLs, Petri Nets, temporal logic, etc.)

Figure 2: Formal Design Analysis Framework [CDDD03]

[Aag01, Zsc04a, Zsc04b] present formal specification languages for non-functional properties of component-based systems. For instance, [Zsc04a] shows a formal specification of timeliness properties of a component-based system, as an example for a formal approach to specifying non-functional properties. For that, the work use extended temporal logic of actions (TLA+) to describe the system. The logic is a temporal logic where states are represented as values assigned to state variables. CQML(Component Quality Modeling Language) [Aag01] is another useful specification language for components QoS properties.

Another work of NFR formalization is [BBF$^+$01]. It present the language NoFun (acronym for NOn-FUNctional) which consists of three different parts. Firstly, software quality characteristics and attributes are defined. Secondly, values are assigned to component quality basic attributes. Thirdly, quality requirements can be stated over components, both context-free and context-dependent. The language contains structuring mechanisms, type definition elements and other constructs that give an appropriate support for defining non-trivial quality models. The work [BBF$^+$01] studies also how these concepts can be mapped to UML using its extension mechanisms.
Recently, [KHP$^+$08] present a work in progress which describes the formal requirement specifications for increasing the safety and reliability of the railway system using Z notation.

3 Design level

The idea of incorporating NFRs with FRs in the design level is not new. A survey of the different works found that most of them use UML with some extensions to add NFRs with the functional requirements models. In the next, we are going to present the different solutions proposed in the literature that try to consider NFRs at the design level.

[SC04] presents a use case and goal-driven approach to integrate functional requirements and NFRs. It propose to associate the NFRs with four use case model elements: actor, use case, actor use case association and the system boundary. Their framework specifies the scope of each type of NFR association through the formalization of NFR scope propagation rules. This take advantage of relationships between the different elements of use case (specialization, generalization, extends, includes). Thus, the solution proposed affirms that NFRs can be integrated at the design level with FRs and can provide better understanding of the requirements model. The authors of [MAB02] propose a new model which identify all the requirements of a system and select from those the quality attributes relevant to the application domain and stakeholders. Then, they propose a template for quality attributes and integrate them with functional requirements using an extended use case diagram. In the same way, [BG06] extends UML use case diagram with some notation to express functional requirements, NFRs and the hazard (area that presents additional challenges to the requirements analyst).

[CL01, NLC00] propose a new strategy that brings NFRs to object-oriented modeling called OONFR (Object-Oriented Non Functional Requirements). This strategy uses as input a Language Extended Lexicon of the Universe of Discourse (LEL of UofD). This latter is composed of entries, which describe symbols of the language of the UofD, through notions and behavioral responses. These entries can be classified as subject, verb, object, and state. The outputs of OONFR strategy is a UML class diagram with indications of what classes, attributes, operations and relationships are responsible for satisfying NFRs. In other words, the class diagram has signals of what classes, attributes, operations, and relationships specialize satisfying strategies and satisfy NFRs. In summary, OONFR strategy consists of the following activities: build the Language Extended Lexicon of Universe of Discourse-NFR (LEL of UofD-NFR), build the scenarios and build the UML class diagram. The authors indicate that the extension of this strategy to other UML artifacts and more automation of the strategy can give it more importance.

Many others works have also studied the integration of NFRs at the design level. For instance, [ZG07] propose a UML profile for modeling design decisions and an associated UML profile for modeling NFRs in a generic way. Hence, the two UML profiles consider design decisions and NFRs as first class elements. This relationship between design decisions and

NFRs is modeled using specialized dependency notations in UML. [SC05] proposes an integrating modeling language by extending UML with the NFR Framework [CNYM00] using the UML Profile. Thus, they define a meta-model to represent the concepts in the NFR Framework and they identify the extension points for integrating the two notations. [TT05] propose a novel framework for integrating NFRs with the UML design of a software system which can be applied during the re-engineering process of legacy systems. [XZRL05] proposes a grouping mechanism to model NFRs in software architectures directly and explicitly. The approach proposes a conceptual architectural design model which views the traditional architecture model for software systems as the first layer. The operations of satisfying NFRs are represented as aspectual components in the second layer. [JÖ2] proposes *UMLsec* which is an extension of UML notation. UMLsec allows to express security relevant information within the diagrams in a system specification. UMLsec is defined in form of a UML profile using the standard UML extension mechanisms. In particular, the associated constraints give criteria to evaluate the security aspects of a system design, by referring to a formal semantics of a simplified fragment of UML. [LBD02] present a modeling language, based on UML, called *SecureUML*. It shows how UML can be used to specify information related to access control in the overall design of an application and how this information can be used to automatically generate complete access control infrastructures. [LBD02] adapt use cases to capture and analyse security requirements. This adaptation is called an Abuse Case Model. An abuse case is defined as a specification of a type of complete interaction between a system and one or more actors, where the results of the interaction are harmful to the system, one of the actors, or one of the stakeholders of the system.

4 Implementation level

A survey of the literature found that current programming languages were not designed with NFRs. However, some works have tried to bridge this gap by proposing new programming styles which process NFRs at the implementation level. In the following, we review various policy mechanisms that can be used to implement NFRs. [Bol00] introduce a new programming style into the existing object-oriented language Java. This new method is called "Constraint and Object Oriented" programming style. So, [Bol00] propose to conservatively extend object-oriented decomposition by letting it affect also operations (methods). Different objects may support different parts of the same operation. The responsibility of defining an operation, in terms of enabling conditions and effects on the state, is distributed over several interacting objects, which act as constraints and express different, partial views about the system behavior. The last revision of Ada language [FL93] is another example of new programming style which takes account different NFRs. Firstly, timeliness requirements will be better served by improved definitions of timeouts and a minimum run-time environment. Secondly, dependability will be assisted by the ability to attach "watchdog" timers to code blocks. Thirdly, security requirements are addressed by well established user authentication, encryption and message certification technologies. Lastly, adaptability will be addressed by the ability to change task priorities dynamically. Exception Mechanism is also another new style programming which is supported by many current programming languages like C++ and Java. This mechanism separates the normal control flow from the exceptional control flow under error conditions. This separation of concerns and centralized exception handling reduce the complexity of programming. Thus, exception mechanism can be viewed as a special form of policy because it provides a mechanism to specify the policies about how to handle faults.

They are also another type of solution proposed by many works that combines rule based techniques and object oriented programming. For example, [Ahm97] present a new language, called R++ rules, which is an extension to C++. R++ rules are triggered automatically upon relevant data change. One important contribution of R++ is that it introduced rule as member of class. R++ Rules are introduced as a natural extension to object-oriented classes, they support inheritance, overriding, and visibility rules. ILOG JRules is another language presented in [Ilo02]. It directly use the Java objects. Jrules are organized into groups called packets. "Packet" is represented as a property of a rule. This language supports temporal reasoning: the "wait" statement is used in the condition part of a rule. Thus, the "wait" statement allows to test if conditions become valid during a designated waiting period.

In summary, the limits of current programming languages for implementing NFRs are well-known. In order to face this problem, we think that the implementation for NFRs shall be separated (in some cases) from the implementation for the functional requirements. We think also that a strong traceability between NFRs and functional requirements can address this problem.

5 Complete approaches

All the above approaches present the way NFRs can be handled within a specific step of the software development process. However, some other approaches cover the whole development process, as presenting in the next.

[BL03] propose a model called FRIDA (From RequIrements to Design using Aspects) in which the key objective is to guide the developer through the basic phases of the software life cycle. FRIDA determine a way to elicit and model separately both functional and non-functional requirements. In order to achieve that, the FRIDA model is founded on the aspect-oriented software development. Aspects are a complementary manner to describe NFRs. In FRIDA, each NFR is represented with one or more aspects and is not restricted to a single level. Functional requirements are restricted to class hierarchies. Checklists are used to refine NFRs at early-stages of the development life cycle. The FRIDA model considers that the identification and resolution of conflicts is as important as the elicitation of NFRs at the early stages of the software development life cycle. For that, FRIDA use a conflicts matrix to automate this process whenever possible. The UML notation and extensions of it are used to determine this model and allow the modeling of both objects and aspects involved in the system. The aspect extraction and the visual modeling of aspects are very important for FRIDA because they allow the creation of models that can be reused in the future. Moreover, connections between diagrams that belong to distinct phases of software development allow a high level of traceability.

[LYL$^+$05] define a coherent goal-driven process applicable at the requirements level as well as the implementation level. This process (called reuse process) uses an asset library to find quality characteristics and apply them to a software functional description. [LYL$^+$05] shows that it is possible to store qualities, retrieve them for reuse, specialize them for different contexts and integrate them with functional descriptions.

The B method [Abr96] is a full formal method which uses set theory notations to specify, design and implement software systems. The most important feature provided by the B method is its ability to stepwise refine specifications. Refinement is a process that transforms an abstract and non-deterministic specification into a concrete and deterministic system that preserves the functionality of the original specification. A key merit of this refinement mechanism is the ability to preserve already proven system properties in higher level models.

Proof obligation are generated from B models. They guarantee correctness and effectiveness of the system development. The B method is supported by tools that guide the developer through the basic phases of the software life cycle. Recently, [HBEE08] present a work in progress using the B method to handle NFRs. It proposes a formal approach that derives design specifications from a set of security requirements modeled using an extension of KAOS for security. The approach provides a secure software engineering methodology that effectively integrates KAOS security extension. It is characterized by the ability to formally build a complete, consistent and clear requirements model with the B method. Figure 3 shows the main steps of this approach.

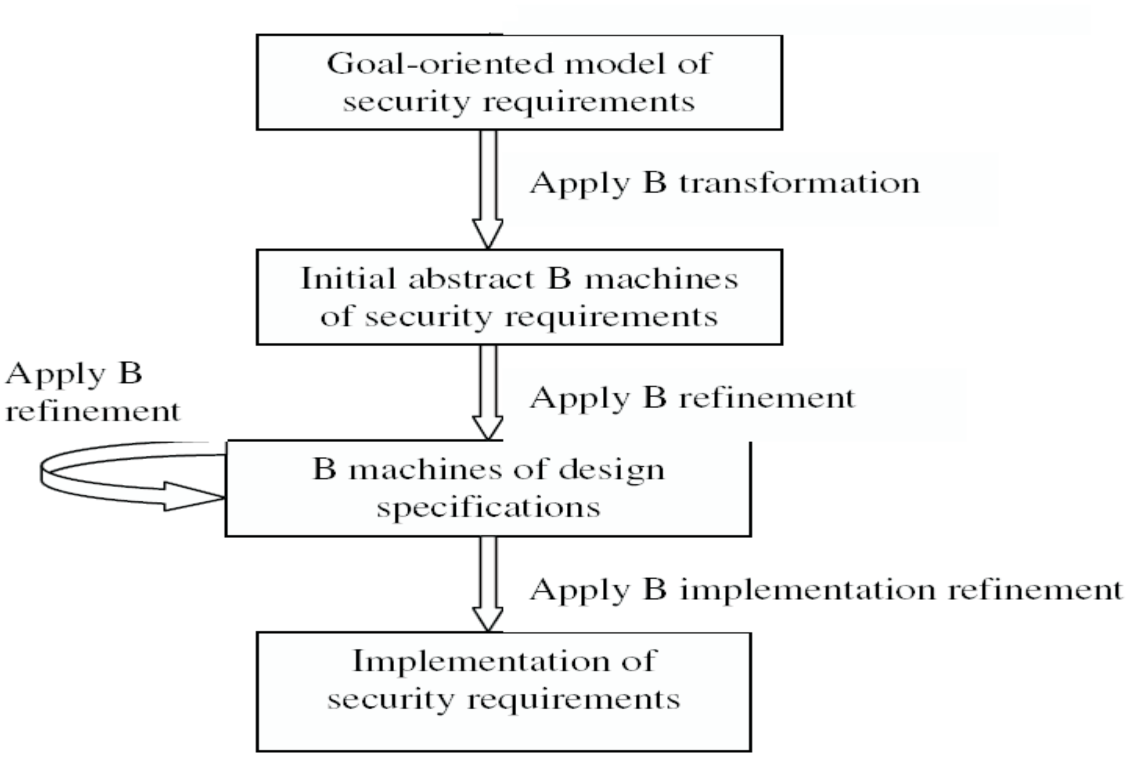
Goal-oriented model of security requirements
Apply B transformation
Initial abstract B machines of security requirements
Apply B refinement
Apply B refinement
B machines of design specifications
Apply B implementation refinement
Implementation of security requirements

Figure 3: Formal Derivation of Security Design Specifications [HBEE08]

The first step of the approach is to transform a KAOS security requirements model to an abstract B model. The second step is to refine the model using B refinement mechanism to generate design specifications and implementation. Thus, the approach shows that KAOS could be extended with an extra step of formality in order to fully implement security requirements. This extension establishes a reasonable bridge between security requirements and design specifications. However, the approach needs to be: (i) demonstrated on a number of different case studies; (ii) generalized to other NFRs.

[MGM03a] proposes an approach that considers security concerns (a type of NFRs) as an integral part of the entire system development process. This approach integrates security and systems engineering using the same concepts and notations, throughout the entire system development process. It falls within the context of the Tropos methodology [CKM01] that considers not only the system functional requirements but also NFRs such as security, reliability, and performance. Tropos is based on the idea of building a model of the system that is incrementally refined and extended from a conceptual level to executable artefacts, by means of a sequence of transformational steps. Tropos adopts the i* modelling framework [Yu95] which uses the concepts of actors, goals, tasks, resources and social dependencies for defining the obligations of actors (dependees) to other actors (dependers). In addition to the graphical representation, Tropos provides a formal specification language called Formal Tropos [FMPT01]. It complements graphical Tropos by extending the Tropos graphical language with a formal specification language. The language offers a textual notation for i* models and allows the description of different elements of the specification in a first order linear-time temporal logic inspired by KAOS [Lam01, Let01] that has formal semantics and is amenable to formal analysis. In addition, Formal Tropos serves also to verify the model of the system by employing formal verification techniques such as model checking to allow for an automatic verification of the system properties. However, both Tropos and Formal Tropos were not conceived with security on mind. [MGMP02, MGM03b] presents extensions to the Tropos ontology to enable it modeling security issues. Concepts and notations such as security constraint, secure entities and secure dependencies are introduced to the existing graphical Tropos and also Formal Tropos grammar was extended to provide formalism for our newly introduced concepts. However, this extension only apply to the first level of the Tropos ontology.

6 Conclusion

Despite the fact that NFRs are very difficult and expensive to deal with, the increasing software complexity and competition in the software industry has highlighted the need to consider NFRs as an integral part of software development process. For that, many works have studied them through the different phases of the software life cycle. Thus, this paper shows that NFRs obviously impinge on all the software development process. However, many problems and shortcomings still remains. Firstly, the informal treatment of NFRS since the first levels of the development chain; i.e. the requirements analysis and specification levels. Secondly, it seems difficult to define one method to cope with all NFRs due to their great diversity. Finally, few research works have focused on NFRs as first class requirements in the development process. To bridge these gaps, researchers face many challenges: (i) extending and relaxing formal methods in order to support the majority of NFRs; (ii) modeling and analyzing functional requirements and NFRs that should be considered separately; (iii) providing methodologies guidance through the whole development process.

References

[Aag01] Jan Oyvind Aagedal. Quality of Service Support in Development of Distributed Systems. *PhD Thesis, University of Oslo*, 2001.

[Abr96] Jean Raymond Abrial. The B-Book: Assigning programs to meanings. *Cambridge University Press*, 1996.

[Ahm97] Amal Ahmed. R++ User Manual for Release 1.1. *AT&T*, 1997.

[BBF$^+$01] Pere Botella, Xavier Burgues, Xavier Franch, Mario Huerta, and Guadalupe Salazar. Modelling Non-Functional Requirements. *Proceedings of Jornadas Ingeniera de Requisitos Aplicados (JIRA), Sevilla, Spain*, 2001.

[BG06] Brian Berenbach and Mark Gall. Toward a Unified Model for Requirements Engineering. *ICGSE*, 0:237–238, 2006.

[BKLW95] Mario Barbacci, Mark H. Klein, Thomas A. Longstaff, and Charles B. Winstock. Quality Attributes. *Technical Report, CMU/SEI-95-TR-021, Software Engineering Institute, Carnegie Mellon University*, 1995.

[BL03] Silvia Bertagnolli and Maria Lisboa. The FRIDA model. In *Analysis of Aspect-Oriented Software (ECOOP 2003)*, July 2003.

[Bol00] Tommaso Bolognesi. Toward Constraint-Object-Oriented Development. *IEEE Trans. Software Eng.*, 26(7):594–616, 2000.

[CDDD03] Kendra Cooper, Lirong Dai, Yi Deng, and Jing Dong. Developing a Formal Design Analysis Framework. In *Software Engineering Research and Practice*, pages 68–73, 2003.

[CKM01] Jaelson Castro, Manuel Kolp, and John Mylopoulos. A Requirements-Driven Development Methodology. In *CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, pages 108–123, 2001.

[CL01] Luiz Marcio Cysneiros and Julio Cesar Sampaio Leite. Using UML to reflect Non-functional Requirements. In *CASCON*, page 2, 2001.

[CNYM00] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. Non-Functional Requirements in Software Engineering. *Boston: Kluwer Academic Publishers*, 2000.

[DKKP03] Jrg Drr, D. Kerkow, Antje Von Knethen, and Barbara Paecha. Eliciting Efficiency Requirements with Use Cases. In *In: Salinesi C, Regnell B, Kamsties E (Hrsg) Proceedings of the 9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'03)*, 2003.

[Fir03] Donald Firesmith. Security Use Cases. *Journal of Object Technology*, 2(3):53–64, 2003.

[FL93] Colin Fidge and Andrew Lister. The Challenges of Non-Functional Computing Requirements. *Seventh Australian Software Engineering Conference (ASWEC93)*, 1993.

[FMPT01] Ariel Fuxman, John Mylopoulos, Marco Pistore, and Paolo Traverso. Model Checking Early Requirements Specifications in Tropos. In *RE '01: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE '01)*, page 174. IEEE Computer Society, 2001.

[Gli07] Martin Glinz. On Non-Functional Requirements. In *15th IEEE International Volume , Issue , 15-19 Oct*, pages 21–26, 2007.

[HBEE08] Riham Hassan, Shawn A. Bohner, Sherif Elkassas, and Mohamed Eltoweissy. Goal-Oriented, B-Based Formal Derivation of Security Design Specifications from Security Requirements. In *ARES*, pages 1443–1450, 2008.

[HJWM06] Chih-Wei Ho, Michael J. Johnson, Laurie Williams, and E. Michael Maximilien. On Agile Performance Requirements Specification and Testing. In *AGILE '06: Proceedings of the conference on AGILE 2006*, pages 47–52. IEEE Computer Society, 2006.

[HP05] Andrea Herrmann and Barbara Paech. Quality Misuse. *REFSQ -International Workshop on Requirements Engineering: Foundation for Software Quality*, 2005.

[Ilo02] Ilog JRules Language Reference,version 3.0. *http://www.ilog.com*, 2002.

[JÖ2] Jan Jürjens. UMLsec: Extending UML for Secure Systems Development. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 412–425, London, UK, 2002. Springer-Verlag.

[KDO07] Mohamad Kassab, M. Daneva, and Olga Ormandjieva. Scope Management of Non-Functional Requirements. In *EUROMICRO '07: Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 409–417. IEEE Computer Society, 2007.

[KHP$^+$08] Miyoung Kang, Dae-Yon Hwang, Junkil Park, Jin-Young Choi, and Jong-Gju Hwang. Formal Requirements Specification for Railway Signaling Systems. In *ABZ 2008, London, UK*, September 16-18, 2008.

[KK99] Haruhiko KAIYA and Kenji KAIJIRI. Refining Behavioral Specification for Satisfying Non-functional Requirements of Stakeholders. In *IEICE transactions on information and systems Vol.E85-D, No.4(20020401)*, pages 623–636, 1999.

[Lam00] Axel Van Lamsweerde. Formal Specification: a Roadmap. In *ICSE - Future of SE Track*, pages 147–159, 2000.

[Lam01] Axel Van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *RE '01: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE '01)*, page 249, Washington, DC, USA, 2001. IEEE Computer Society.

[Lam04] Axel Van Lamsweerde. Elaborating Security Requirements by Construction of Intentional Anti-Models. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 148–157, Washington, DC, USA, 2004. IEEE Computer Society.

[LBD02] Torsten Lodderstedt, David A. Basin, and Jürgen Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 426–441, London, UK, 2002. Springer-Verlag.

[Let01] Emmanuel Letier. Reasoning About Agents in Goal-Oriented Requirements Engineering. *Ph.D. Thesis, ftp://ftp.info.ucl.ac.be/pub/thesis/letier.pdf*, 2001.

[LYL$^+$05] Julio Cesar Sampaio Prado Leite, Yijun Yu, Lin Liu, Eric S.K. Yu, and John Mylopoulos. Quality-Based Software Reuse. volume 3520, pages 535–550, 2005.

[MAB02] Ana Moreira, Joao Araujo, and Isabel Brito. Crosscutting Quality Attributes for Requirements Engineering. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering, Italy*, pages 167–174, 2002.

[MGM03a] Haralambos Mouratidis, Paolo Giorgini, and Gordon Manson. Integrating Security and Systems Engineering: Towards the Modelling of Secure Information Systems. In *Advanced Information Systems Engineering, 15th International Conference, CAiSE 2003, Klagenfurt, Austria*, volume 2681, page 6378, June 2003.

[MGM03b] Haralambos Mouratidis, Paolo Giorgini, and Gordon A. Manson. An Ontology for Modelling Security: The Tropos Approach. In *KES*, pages 1387–1394, 2003.

[MGMP02] Haralambos Mouratidis, Paolo Giorgini, Gordon Manson, and Lan Philp. A Natural Extension of Tropos Methodology for Modelling Security. In *In the Proceedings of the Agent Oriented Methodologies Workshop (OOPSLA 2002), Seattle, USA*, November 2002.

[NLC00] Jaime De Melo Sabat Neto, Julio Cesar Sampaio Leite, and Luiz Marcio Cysneiros. Non-Functional Requirements for Object-Oriented Modeling. In *WER*, pages 109–125, 2000.

[PEM03] Frauke Paetsch, Armin Eberlein, and Frank Maurer. Requirements Engineering and Agile Software Development. In *WETICE '03: Proceedings of the Twelfth International Workshop on Enabling Technologies*, page 308. IEEE Computer Society, 2003.

[SC04] Sam Supakkul and Lawrence Chung. Integrating FRs and NFRs: A Use Case and Goal Driven Approach. *Proc. SERA 04*, pages 30–37, 2004.

[SC05] Sam Supakkul and Lawrence Chung. A UML Profile for Goal-Oriented and Use Case-Driven Representation of NFRs and FRs. In *SERA*, pages 112–121, 2005.

[SO05] Guttorm Sindre and L. Opdahl. Eliciting Security Requirements with Misuse Cases. *Requir. Eng.*, 10(1):34–44, 2005.

[TT05] Subrina A. Tonu and Ladan Tahvildari. Towards a Framework to Incorporate NFRs into UML Models. In *Proceedings of IEEE WCRE Workshop on Reverse Engineering to Requirements (RETR), Pittsburgh, Pennsylvania, USA*, pages 13–18, November 2005.

[XZRL05] Lihua Xu, Hadar Ziv, Debra Richardson, and Zhixiong Liu. Towards Modeling Non-Functional Requirements in Software Architecture. 2005.

[Yu95] Eric Yu. Modelling Strategic Relationships for Process Reengineering. *Ph.D. Thesis, Department of Computer Science, University of Toronto, Canada*, 1995.

[Zav97] Pamela Zave. Classification of Research Efforts in Requirements Engineering. *ACM Comput. Surv.*, 29(4):315–321, 1997.

[ZG07] Liming Zhu and Ian Gorton. UML Profiles for Design Decisions and Non-Functional Requirements. In *2ndIntl. Workshop on SHAringand Reusing architectural Knowledge*, 2007.

[Zsc04a] Steffen Zschaler. Formal Specification of Non-functional Properties of Component-Based Software. In Jean-Michel Bruel, Geri Georg, Heinrich Hussmann, Ileana Ober, Christoph Pohl, Jon Whittle, and Steffen Zschaler, editors, *Workshop on Models for Non-functional Aspects of Component-Based Software (NfC'04) at UML conference 2004*, September 2004. Technical Report TUD-FI04-12 Sept.2004 at Technische Universität Dresden.

[Zsc04b] Steffen Zschaler. Towards a Semantic Framework for Non-functional Specifications of Component-Based Systems. In Ralf Steinmetz and Andreas Mauthe, editors, *Proc. EUROMICRO Conf. 2004*, Rennes, France, September 2004. IEEE Computer Society.