PhD Theses in Experimental Software Engineering

Vol. 34

Jörg Dörr

Elicitation of a Complete Set of Non-Functional Requirements

FRAUNHOFER VERLAG

PhD Theses in Experimental Software Engineering

Volume 34

Zugl.: Kaiserslautern, Univ., Diss., 2010

Printing:
Mediendienstleistungen des
Fraunhofer-Informationszentrum Raum und Bau IRB, Stuttgart

Printed on acid-free and chlorine-free bleached paper.

ISBN 978-3-8396-0261-4
Fraunhofer-Informationszentrum Raum und Bau IRB
Postfach 800469, 70504 Stuttgart
Nobelstraße 12, 70569 Stuttgart
Telefon +49 711 970-2500
Telefax +49 711 970-2508
E-Mail verlag@fraunhofer.de
URL http://verlag.fraunhofer.de

Elicitation of a Complete Set of Non-Functional Requirements

Beim Fachbereich Informatik
der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation
von

Dipl.-Inform. Jörg Dörr

Fraunhofer-Institut für Experimentelles Software Engineering
(Fraunhofer IESE)
Kaiserslautern

Berichterstatter:	Prof. Dr. Dr. h.c. Dieter Rombach Prof. Dr. Barbara Paech
Dekan:	Prof. Dr. Karsten Berns
Tag der Wissenschaftlichen Aussprache:	15.07.2010

D 386

Acknowledgement

First of all, I would like to thank Prof. Barbara Paech for encouraging me to pursue the topic of non-functional requirements. I soon realized the importance of the topic for industry as well as for academia. In the ITEA-funded Empress project, I had the unique opportunity to cooperate with outstanding people and develop the basis for the method of this thesis. My thanks go to everyone involved in the Empress project!

Second, I would like to thank my two supervisors Prof. Dieter Rombach and Prof. Barbara Paech for their valuable advice and guidance during this thesis! I would like to thank Prof. Dieter Rombach for creating an environment that made it possible for me to pursue my PhD. The unique environment at Fraunhofer IESE made it possible to develop this methodology, but it particularly enabled the validation in eight industrial case studies.

Third, I would like to thank all former and current colleagues at Fraunhofer IESE for many fruitful discussions and honest feedback, especially the members of the QSD and later on RUE department! Also, I would like to thank Dr. Dirk Muthig for his advice on focusing my thesis topic.

During the development of this thesis, I cooperated with many outstanding people in the application of the methodology. So, fourth, I would like to thank all people at IESE who jointly applied the NFR methodology with me in the case studies, i.e., Sebastian Adam, Michael Eisenbarth, Daniel Kerkow, Tom Koenig, Dennis Landmann, Thomas Olsson, and Antje von Knethen (in alphabetical order). Furthermore, I would like to thank all the people who performed the method without me in the case studies, i.e., Sebastian Adam, Anne Gross, Tom Koenig, and Thomas Olsson (in alphabetical order).

Fifth, I would like to thank all participants of the working group "Non-functional requirements" of the German Computer Society (GI). I want to thank them for the exchange of experiences and for the valuable discussions we had at all the meetings.

Sixth, I would like to thank Martha Hernandez and Michael Yaco who supported the tool development with their outstanding work as part of their student research assistance jobs.

Seventh, I would like to thank my dear colleagues and friends with whom I studied, work and celebrate, i.e., Ralf Carbon, Christian Denger, Michael Eisenbarth, Tom Koenig, Marcus Trapp, and Mario Trapp. With them, study and work were and are always enjoyable and professional at the same time. I want to thank them for their support!

Last, but far from least, I want to thank my wife Anette and my children Jannik and Amelie! Their love, encouragement, support, and understanding made it possible to spend evenings, weekends, and vacation time on this thesis instead of on family activities.

Abstract

Requirements engineering is the first activity in engineering a software-based product. Making mistakes in such an early phase has a strong impact on all subsequent software development phases. Especially non-functional requirements (NFRs) play an important role for the success of a project or product. In today's practice, essential information on a system's NFRs have often not been elicited properly and are thus incomplete. As a result, architectures have to change in late development phases, which leads to increased project or platform development costs and increased time to market. Alternatively, missing NFRs are not incorporated into the product in later phases, leading to low product quality.

This thesis addresses the topic of complete NFR elicitation. It focuses on NFR elicitation and specification for software-based, interactive systems. Current state-of-the-practice and state-of-the-art approaches treat NFRs in parallel to functional specifications. Neither the state-of-the-practice nor the state-of-the-art approaches offer a possibility to judge the completeness of the NFR elicitation.

The NFR methodology described in this thesis provides a systematic approach for the elicitation, analysis and specification of a complete set of NFRs. To achieve this, the NFR methodology contributes an algorithm to elicit NFRs, which takes quality model information as input and systematically processes specific elements of the functional specification. The algorithm is based on a requirements taxonomy and a metamodel relating functional, non-functional, and architectural concepts. This thesis also provides complete and detailed process guidance on how to use the elicitation algorithm for NFR elicitation and specification. Checklists and tool support are used to support and partially automate the NFR methodology.

The NFR methodology was evaluated in a series of eight, mainly industrial, case studies. The evaluation showed that the NFR methodology is feasible and results in a more complete set of NFRs. The ratio of newly identified, important NFRs ranges from over 100% to 622%. Moreover, the evaluation showed that the estimated rework effort in the following project or platform development phases is significantly reduced. In two case studies, the NFR methodology application resulted in an ROI > 2 and an ROI > 17, respectively. As a positive side effect, the evaluation showed that consequently using the NFR methodology can lead to measurable NFRs. Rates of 95.5% and even up to 100% of NFRs being measurable were achieved in the case studies.

Table of Contents

Acknowledgement ... iii
Abstract ... v
Table of Contents ... vii
List of Figures ... ix
List of Tables ... xi

1 Introduction ... 1
1.1 The Role of Non-Functional Requirements ... 1
1.2 Contribution and Research Hypotheses ... 5
1.3 Research Approach ... 7
1.4 Outline ... 10

2 Classifications, Definitions and Quality Models ... 12
2.1 State of the Practice and State of the Art ... 12
2.1.1 Existing Definitions for NFRs ... 12
2.1.2 Existing Quality Models ... 17
2.2 Taxonomy, Definitions, Metamodel & Quality Model ... 21
2.2.1 Requirements Taxonomy ... 21
2.2.2 Metamodel ... 23
2.2.3 Quality Models ... 38
2.2.4 Definition of Completeness ... 47

3 Specification of FRs and NFRs ... 50
3.1 State of the Practice and State of the Art ... 50
3.1.1 State of the Practice in Specifying FRs and NFRs ... 50
3.1.2 State of the Art in Specifying FRs and NFRs ... 52
3.1.3 Relationship Between FRs and NFRs ... 59
3.2 Integrated FR and NFR Specification ... 65
3.2.1 Relationship of Functional Elements to NFRs ... 65
3.2.2 The Challenge of Integrated Specification ... 70
3.2.3 Resulting Locations for NFR Specification ... 74

4 Elicitation of NFRs ... 76
4.1 Main Difference to Existing Approaches ... 76
4.2 The Elicitation Algorithm ... 78
4.3 The Role of Checklists as an Elicitation Aid ... 82
4.4 The Role of Reference Checklists and Templates ... 84

5 The NFR Elicitation and Specification Process ... 89
5.1 Overview of the NFR Process ... 89
5.2 Process Activity P1.1: Prioritize QAs ... 91
5.3 Process Activity P1.2: Tailor Quality Models ... 92
5.4 Process Activity P1.3: Identify Possible Dependencies ... 97
5.5 Process Activity P1.4: Derive Checklist and Template ... 99

5.6 Process Activity P2.1: Elicit and Specify NFRs 104
5.7 Process Activity P2.2: Identify NFR Dependencies 110
5.8 Learning from Project Experience 113

6 Increasing the Method's Efficiency 116
6.1 Focusing the Effort 116
6.1.1 Trading off the Quality Scope 117
6.1.2 Trading off the Functional Scope 118
6.1.3 Trading off the Dependencies 119
6.1.4 Performing the NFR Methodology Iteratively 122
6.1.5 Change Management View 124
6.2 Tool Support 125
6.2.1 The Checklist Generation Tool 127
6.2.2 The Elicitation Guide Tool 131

7 Validation 135
7.1 Introduction 136
7.2 Case Study Contexts 137
7.3 Validation Goals 141
7.4 Case Study Data and Results 143
7.4.1 Feasibility 144
7.4.2 Completeness 147
7.4.3 Effort 151
7.4.4 Further Qualitative or Quantitative Observations 154
7.5 Summary of Validation Results and Discussion 157
7.5.1 Summary of Results 157
7.5.2 Threats to Validity 158
7.5.3 Open Questions and Implications 161

8 Summary 163
8.1 Results and Contribution 163
8.2 Method Limitations and Future Work 166
8.3 Concluding Remarks 168

References 170

Appendix A: Example Quality Models 181

Appendix B: Template for an Integrated Specification 188

Appendix C: Effort Data for Case Studies 195

Appendix D: Detailed Expert Estimate in EMERGE 196

Appendix E: List of Abbreviations 197

Lebenslauf 199

List of Figures

Figure 1:	Specified NFRs as enabler, missing NFRs as threat	2
Figure 2:	Quality trade-offs in SW architecture phase	3
Figure 3:	Research approach	8
Figure 4:	Faceted classification of requirements (figure taken from [Gli07])	16
Figure 5:	ISO 9126 view on qualities	18
Figure 6:	ISO 9126 quality model for external and internal attributes	19
Figure 7:	ISO 9126 quality model for quality in use	19
Figure 8:	Requirements taxonomy	21
Figure 9:	Simplified version of the metamodel	24
Figure 10:	Functional elements of the metamodel	28
Figure 11:	Quality related elements of the metamodel	31
Figure 12:	Complete metamodel for the NFR methodology	35
Figure 13:	Typical set of HLQAs	41
Figure 14:	Example of $QM\_Ref_{Efficiency}$	43
Figure 15:	Example dependency matrix for $QM\_Ref_{Efficiency}$	46
Figure 16:	Example illustration for completeness definition	48
Figure 17:	Recommended sections of the Volere specification template	51
Figure 18:	Example of a SIG as used in the NFR Framework (figure taken from [CNY+99])	54
Figure 19:	Example of a MOQARE misuse tree showing the main MOQARE concepts (figure taken from [HKD07])	56
Figure 20:	Concepts and examples of the UMD approach (figure taken from [BDA04])	57
Figure 21:	Example of annotating an ER model with quality attributes (figure taken from [CL99])	62
Figure 22:	Non-integrated specification of FR and NFR in state of the practice and state of the art	64
Figure 23:	Mapping of functional metamodel elements to the TORE decision model	66
Figure 24:	Illustration of the problem of Primary and Secondary Information Place	72
Figure 25:	Example of annotation of User Task NFRs in a UC diagram	75
Figure 26:	Items to be compared in elicitation algorithm, illustrated in a comparison matrix	78
Figure 27:	Simplified version of the elicitation algorithm	79
Figure 28:	The elicitation algorithm	80
Figure 29:	The different steps of the algorithm visualized in the comparison matrix	82

Figure 30: Checklists as mediator between quality models and NFRs 84
Figure 31: Reference checklist for $QM\_Ref_{Efficiency}$ 87
Figure 32: The NFR process 90
Figure 33: Tailored efficiency model: $QM\_InScope_{Efficiency}$ 96
Figure 34: Intermediate result of P1.3 98
Figure 35: Final dependency matrix for $QM\_InScope_{Efficiency}$ 99
Figure 36: Project-specific checklist for $QM\_InScope_{Efficiency}$ 103
Figure 37: Physical components in the X project 108
Figure 38: Target UCs for NFR elicitation in the X project 108
Figure 39: Use Case "Handle Alarm" in the X project 109
Figure 40: Added chapter for general NFR 110
Figure 41: Comparison matrix after first iteration 122
Figure 42: Exemplified instance of NFR algorithm for second iteration 124
Figure 43: Tool support for the NFR process 126
Figure 44: Basic design of the Checklist Generation Tool 127
Figure 45: Main-Screen of Checklist Generation Tool 128
Figure 46: Excerpt of reference checklist for $QM\_Ref_{Efficiency}$ generated with additional features 130
Figure 47: Basic design of the Elicitation Guide Tool 131
Figure 48: Start screen of the Elicitation Guide Tool 132
Figure 49: Horizontal scope selection in the Elicitation Guide Tool 133
Figure 50: Elicitation support in the Elicitation Guide Tool 134
Figure 51: Quality attribute occurrence in case studies 139

List of Tables

Table 1: Existing definitions for the term non-functional requirements 14
Table 2: Comparison of concepts with state-of-the-art approaches 58
Table 3: Minimal set of conceptual elements required in templates 68
Table 4: Extension of set of conceptual elements in templates 69
Table 5: Sentence pattern for creating checklist advices 86
Table 6: Classification of QAs 95
Table 7: Change of requirements document template based on QAs 100
Table 8: Instantiated sentence pattern for checklist advices 101
Table 9: Specifying NFRs in an integrated requirements specification 106
Table 10: Goals of the focus areas 117
Table 11: Overview of the eight case studies 137
Table 12: Summary of results for the feasibility goal 144
Table 13: Summary of results for the completeness goal 148
Table 14: Summary of results for the effort goal 151
Table 15: Effort spent and automation potential 152
Table 16: Measurable NFRs in FIN 154
Table 17: Measurable NFRs in Empress 155
Table 18: Measurable NFRs in EMERGE 156

1 Introduction

"The hardest single part of building a software system is deciding precisely what to build" [Bro87].

Starting with a motivation for the topic of this thesis, this chapter addresses the contributions of this thesis and describes its structure.

1.1 The Role of Non-Functional Requirements

Requirements engineering is the first activity in engineering a software-based product. Therefore, it sets the foundation for the customers' and end users' perception of the final product. Without a doubt, making wrong decisions during this phase has a strong impact on the final product. Whereas functional requirements deal with the functionality the final product will provide, the non-functional requirements (NFRs) determine to a large extent the product's quality, such as the product's performance, usability, or maintainability. Specifying NFR is a key enabler for various subsequent software development activities. Missing NFRs are a major threat to project and product success. Figure 1 gives an overview of the potentials and threats when dealing with NFRs. [BH96], for example, states that "without a well-defined set of quality-attribute requirements, software projects are vulnerable to failure". [CL01c] states that not eliciting NFRs has led to a series of historic failures, including deactivation of a system right after its deployment. [BLF02] and [CL01b] give the example of the London ambulance system (referring to [FD96]) where the deactivation of the system right after its deployment was strongly influenced by non-compliance with NFRs. In that context, [BLF02] states that "we are surprised to verify that non-functional requirements played a very extensive role in the downfall of the system". [CL01c] also state that the lack of integration of NFRs can result in projects that will take more time to be concluded as well as higher maintenance costs.

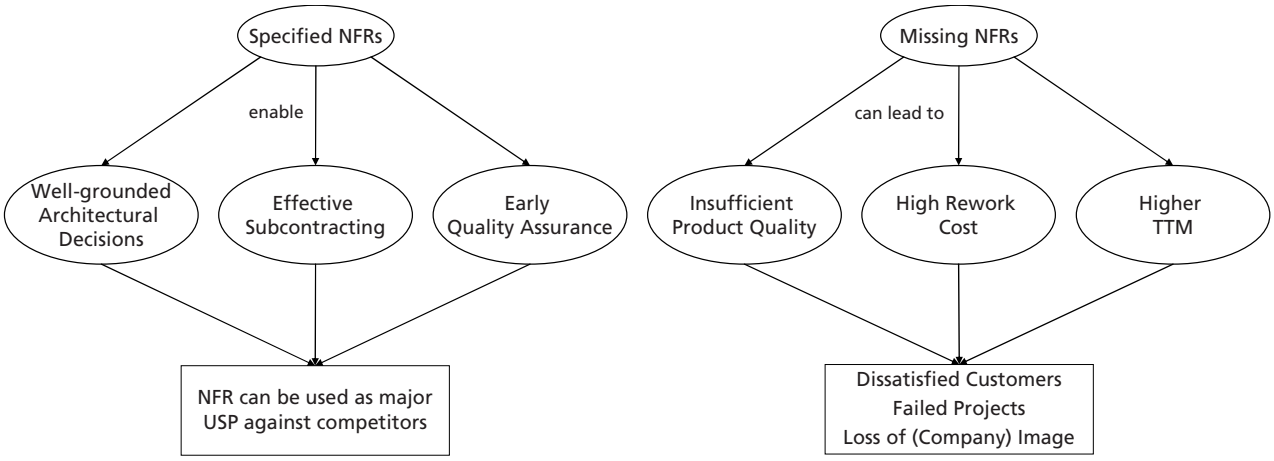


Figure 1: Specified NFRs as enabler, missing NFRs as threat

In the development of software-based systems, NFRs enable

- well-grounded architectural decisions: Almost all typical quality requirements (usability, maintainability, performance, etc.) impact the architectural decisions.
- effective subcontracting: In order to get a suitable product or component from a subcontractor, one has to specify the required quality properties in addition to the required functionality. If only functional requirements are specified, the contractor might be in a situation that the subcontractor delivers a product that fulfills the contract but is not usable for the contractor due to insufficient quality.
- early quality assurance: When measurable NFRs are in place, quality assurance can start immediately after the requirements phase.

When eliciting NFRs systematically, companies can produce high-quality products and the quality characteristics of the products can be used as unique selling propositions to distinguish their products from competitor products. On the other hand, missing NFRs can lead to

- insufficient product quality: If the project responsibles realize that the product does not fulfill the quality needs but it is too late to change the product, the product will be delivered with low quality.
- high rework cost & higher time to market: If the project responsibles realize that the product does not fulfill the quality needs and the project decides to rework the product to match the quality expectations, the project is consuming more effort than planned and the time to deliver the product is postponed.

Impact of NFRs on Architecture

The impact of NFRs on the architecture as the very next phase is the major motivator in the course of this thesis: In today's practice, architectures often have to change in late development phases, which leads to increased project or platform development cost and increased time to market. Typically, there are two reasons for that: The architecture was not designed properly, or essential information on the system's requirements was not elicited properly or changed. Requirements that have such a strong impact on the architecture are typically non-functional requirements (NFR), not forgotten functionality. One major task in architecting a solution for the given requirements is trading off the quality attributes of the architecture (see Figure 2). [Ebe98] states that "The degree of achieving NFRs is predominantely determined during architectural design." [CL01c] performed three case studies. Their approach to elicit NFRs more comprehensively had a strong impact on the functional model: 46% of the existing classes had to be modified, 45% new operations were inserted into class diagrams, and 35% new attributes had to be inserted.

Already [Rom85] stated that "design complexity is also determined by the nature of the NFR." Especially in the context of COTS, [Beu00] argues for the systematic treatment of NFRs as a basis for making architectural decisions. [Pas03] uses trade-off analysis to select architectural decisions based on NFRs. [CL01c] states that NFRs such as safety, performance, accuracy, and others frequently demand that the design is studied carefully in order to fulfill the NFRs to a defined extent. The importance of NFRs for architectural decisions is also emphasized in [CNY95a] and architecture evaluation methods like the Architecture Tradeoff Analysis Method (ATAM) [KKC00], [KBK+99].

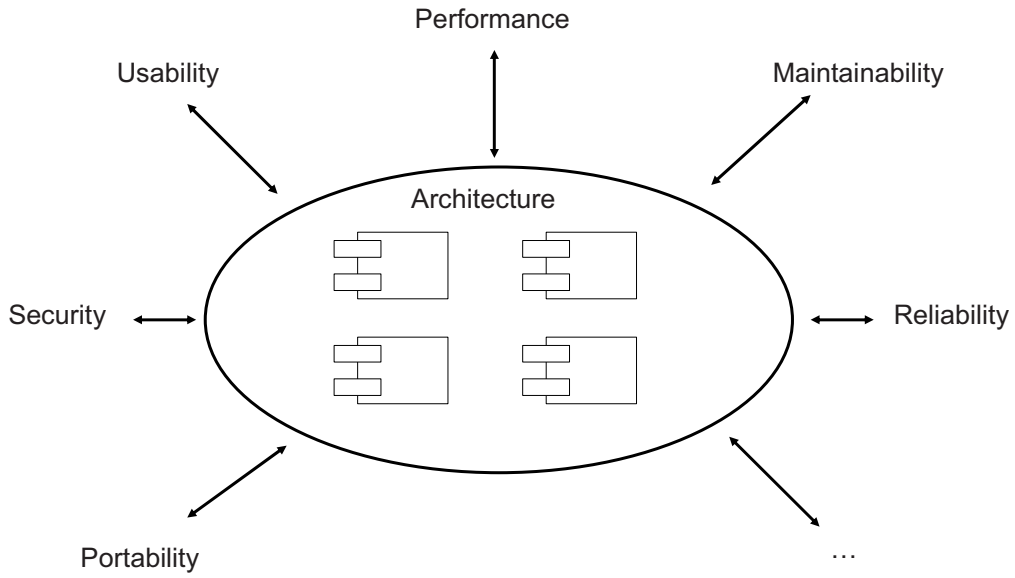


Figure 2: Quality trade-offs in SW architecture phase

[BB02] state that there are often many ways of meeting a FR. NFRs provide guidance for differentiating between these solutions. Furthermore, [BB02] call NFRs preferences, indicating that when given a choice between solutions, one would select one based on these preferences. Also, [Rom85] states that NFRs restrict the types of solutions one might consider.

Role of NFRs in State of the Art and State of the Practice

During the last decades, the state of the art has given much thought to how to elicit, specify, and validate functional requirements. Concerning non-functional requirements, already [BBL76] gives examples of the negative effect of neglecting non-functional properties. According to [BLF02], the role of non-functional requirements in RE was first brought up by [YZC+84] in 1984. Since then, academia confirmed the relevance of NFRs and their huge impact on software engineering: [LWE01], for example, states:

- Neglecting NFRs is among the top six risks in requirements engineering
- Neglecting NFRs (performance) is often worse than forgetting a stakeholder
- Neglecting NFRs (performance) often leads to the need to re-architect
- The NFRs, in their approach called (quality) attribute requirements, "are the heart and soul of why your customers will value your software."

[Rom85] calls the formalization of NFRs one of the main issues in requirements engineering. [CL01c] point out that NFRs are the most expensive and difficult ones to correct [Bro87], [CL99], [Dav93]. [CL01b] states that during the 2001 ICSE, Mantis Chen from ACD System presented the three most important aspects for a software from the stakeholders' point of view and the three most important ones from the developers' point of view. All six were non-functional requirements.

Still, few state-of-the-art approaches targeting NFRs have emerged, and almost none of them has found its way to industry. None of them gives a satisfactory answer to how to achieve a complete set of NFRs, many of the existing approaches target at one specific quality aspect. Till today, NFRs are often not treated systematically. In today's system development, typically the state of the practice foresees some sections on NFRs in requirements documents, but they are mostly filled in unsystematically, leading to low quality of the documentation. [BGR09] claim as a result of a recent industrial study "we did not encounter QR-specific elicitation, documentation or analysis". With QR they refer to quality requirements as synonym to NFRs. The state of the art focuses on NFR identification more or less independent of the functional specification. NFR identification is done in parallel to the functional specification. Efforts to check the elicited set of NFRs with early quality assurance can reveal the prob-

lem that important NFRs might be missing, but do not provide a solution for how to find a complete set of architecture-relevant NFRs.

NFRs are Not Easy to Elicit

This is partly due to the fact that NFRs are not easy to elicit. [Beu00] states: "Non-functional software requirements are notorious for being difficult to elicit, express, quantify and test." [PK04a] and [Jac99] also state that NFR are difficult to elicit and specify. This is due to several reasons. One reason is the effort for dealing with NFRs: [CL01c] and [CL01b] state that NFRs are among the most expensive and difficult to deal with. Some authors point to the "hidden" nature of NFRs: [CL01b] states that NFRs are difficult to elicit due to the very abstract nature of NFRs and because quality aspects, in spite of their importance, are usually hidden in everyone's mind. [LL98] states that "It seems to be inherent into human requirements negotiation that it is easier to state requirements in terms of concrete functional requirements and architectural requirements than in terms of quality attributes. Thus, it is important to better understand and cope with this phenomenon." Also [Gil07] states that "We are not skilled at communicating the 'How well' Product Qualities", referring to NFRs. A third reason is that NFRs are often hard to specify in a measurable manner: [Ebe98] states that often NFRs "are left out because they are difficult to specify with measurable acceptance criteria, thus later leading to discussions during acceptance and handover on exactly these areas."

1.2 Contribution and Research Hypotheses

On the one hand, NFRs have a strong impact on product quality and influence especially the architecture phase. On the other hand, there is the impression that NFRs are not easy to elicit. In 2001, [CL01b] stated that many approaches for NFRs have almost added nothing to the aspect of how to elicit NFRs. This is partly due to the fact that several approaches are strongly dependent on the person eliciting the NFRs. It is natural to be in some sense dependent on the customers and users who will provide the concrete NFRs. The degree to which one is dependent on the requirements analyst who elicits and specifies the NFRs with the customer also adds to this impression. Therefore, requirements analysts need an approach for NFR elicitation that is as systematic, repeatable, and as person-independent as possible. Furthermore, experience with quality characteristics from previous NFR elicitations can help to make the NFR elicitation more complete. To arrive at a repeatable and algorithmic elicitation of requirements, one needs input that can be processed systematically. [CNY+99] made the statement that "…NFRs … often are retrofitted late in the development process, or pursued in parallel with, but separately from, functional design." [CL01c] sees that there is still a gap regarding the integration of NFRs into functional requirements, especially conceptual models. [PK04a] clearly state the necessity to identify NFRs relative to functional requirements and architectural requirements.

[Bus09] stated "If a nonfunctional requirement can't be tied to functional requirements, it isn't needed."

Key Idea of this Thesis

The key idea of this thesis is the systematic elicitation of NFRs, taking specific elements of the functional specification as input and algorithmically processing the functional specification elements. The claim is that a complete set of NFRs can only be achieved if the NFRs are elicited systematically on the set of functional requirements. By systematically processing the elements of the functional specification, the process becomes repeatable and controllable, which is the main driver for increasing the confidence that all important NFRs have been identified. Furthermore, experience-based quality models are used to provide a classified hierarchy of quality aspects. In the systematic NFR elicitation, the functional specification elements are checked against these quality aspects.

Research Objectives

The general goal of this thesis is to provide a systematic approach (in the following called "NFR methodology") for the elicitation, analysis, and specification of a complete set of NFRs. As a positive side-effect, we expect the set of NFRs to be conflict-free and each NFR to be measurable.

To achieve this, the contribution of this thesis includes:

- a classification of functional, non-functional, and architectural concepts and their relationships,
- a representation of quality models that capture hierarchical, classified quality attributes and their dependencies,
- a guideline for the integrated specification of functional and non-functional requirements,
- an algorithm for eliciting NFRs based on functional specification elements and quality attributes,
- checklists and tools to support and partially automate the NFR methodology,
- means for identifying conflicts between NFRs,
- means for focusing the effort for NFR elicitation on the critical qualities and functionalities, and
- complete and detailed process guidance for using the algorithm, all artifacts, and the tool support for NFR elicitation and specification.

Scope

The scope of this thesis is defined as follows:

- Focus on NFR elicitation and specification for software-based, interactive systems.
- Provide support not for only one specific quality attribute but for as many quality attributes as possible, i.e., without restriction to one specific quality attribute (such as efficiency or usability).

Hypotheses

Concerning the benefit and applicability of the methodology developed in this thesis, there are three main hypotheses:

H1 – Feasibility: The elements of the NFR methodology are feasible, i.e., the artifacts can be created for real-life examples and the process activities can be applied by averagely trained personnel.

H2 – Completeness: The method results in a (more) complete set of NFRs. About 20% more critical NFRs are identified compared to the state of the practice. This is expected because the NFRs are elicited using experience-based quality models in a repeatable process.

H3 – Effort: The estimated rework effort in the subsequent project or platform development phases is reduced: The estimated saved rework effort for found NFRs is at least twice the effort spent on systematically eliciting the NFRs.

In the course of this thesis, these three hypotheses will be further refined and evaluated in case studies.

1.3 Research Approach

The scientific approach to realizing the aforementioned research objectives is the experimental software engineering paradigm [Bas93]. The scientific method is used to observe the world, build a new model, and validate it with regard to explicitly stated hypotheses. More specifically, for this thesis, we used a combination of the engineering method and the empirical method:

- Using the engineering method means that we observe existing solutions, develop new and hopefully better solutions, and repeat this process until no further improvements seem to be possible.
- Using the empirical method means that we take a newly developed model and validate it by means of case studies and experiments.

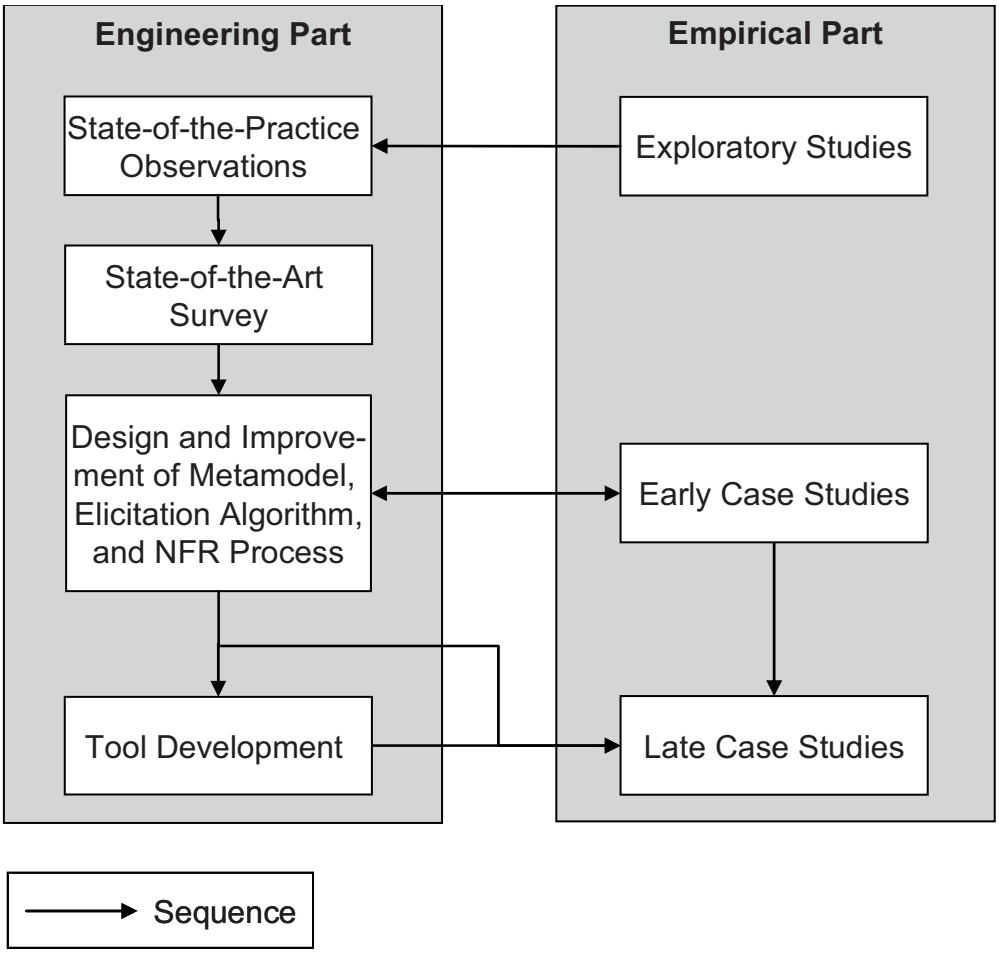


Figure 3: Research approach

The general research approach is depicted in Figure 3. In the following, we explain which elements of the research approach belong to the engineering method and which belong to the empirical method. We experienced the problem of incomplete NFRs in exploratory studies with industrial partners, mainly during the EMPRESS project [Loo03].

Using the Engineering Method

Based on this, we observed the state of the practice in our industry projects in the last eight years, confirmed the situation that NFRs were insufficiently elicited and specified and we experienced the resulting negative impact on the later development phases. The same situation is also reported by other sources, e.g., [BBL76], [CL01c], [BLF02], [LWE01].

We surveyed the state-of-the-art approaches with regard to NFR elicitation, specification, and modelling and found no approach that enables systematic elicitation of NFRs to receive a complete set of NFRs. Additionally, most approaches were not feasible for industrial application or few statements about the effort required to perform the approaches existed. Systematic treatment of NFRs in these approaches was mainly based on using quality models, which is also an important concept in our NFR methodology.

As one of the major problems with the state-of-the-practice and state-of-the-art approaches we identified the fact that they lack a formal basis for judging when the NFR elicitation is complete. Based on a literature survey on existing non-functional requirements terminology, a precise metamodel was defined that states functional, and non-functional conceptual elements and their relationships, which was a key enabler for the subsequent research activities that led to the methodological innovations. A second element is the algorithm developed for systematic NFR elicitation. The metamodel served as the basis for the key idea of algorithmically processing certain types of functional elements for NFR elicitation. Third, the places for documenting the resulting NFRs needed to be specified in accordance with the model. The individual elements of the NFR methodology were aligned into a coherent NFR elicitation and specification process. In order to make the methodology more efficient, two tools were developed. One tool supports the NFR elicitation and one the preparation of NFR elicitation (i.e., the generation of checklists for NFR elicitation).

Using the Empirical Method

Based on the experience gathered in the exploratory studies, the researched NFR methodology with all the new elements was engineered as part of the engineering method. This NFR methodology was then validated in eight mainly industrial settings to provide insights with regard to the method's feasibility, NFR completeness, and the effort needed. The usage of the NFR methodology in early case studies revealed that the metamodel was incomplete, as some NFRs were hard to elicit and classify. Therefore, this information was fed back to the engineering part of the research approach to further improve the model elements, i.e., the metamodel and the elements using the metamodel were revised. Furthermore, by repeating the NFR methodology in the case studies, the potential for tool support became apparent. This was also fed back to the engineering method part of the research approach. The revised version of the NFR methodological elements as well as part of the tool support were then validated in the late case studies.

Additional Informal Validation

Besides the more formal validation with case studies, informal validation of the methodological elements also took place: First, in order to receive feedback from the academic communities, we published the results at various workshops, conferences, journals, and in a book [KKD03], [PvKD+03], [DKvK+03a], [DKvK+03b], [DOS04], [KDP+04], [DKK+05], [DKK+06], [AD07a], [AD07b], [HKD07], [ADB+08], [ARD09], and [MRS+09]. Second, we established a working group of the German Computer Society (Gesellschaft für Informatik) [Doe09] on the topic of NFRs in order to discuss and informally validate the methodological elements with people from industry and academia. Third, to further enhance the exchange of ideas on dealing with NFRs, we organized two workshops on this topic at the German Software Engineering conference [Wor07], [Wor08]. A third one has already been accepted at the German Software Engineering conference for 2010 [Wor10].

1.4 Outline

This thesis is structured as follows:

Chapter 2 lays the foundation for the NFR methodology. It starts with a description of the current state of the practice and state of the art with regard to non-functional requirements definitions, classifications, and existing quality models such as ISO 9126 [Iso01] as a basis for non-functional requirements elicitation. The different kinds of requirements are arranged in a requirements taxonomy. A metamodel that shows the relationship between quality, functional, and non-functional concepts is introduced. The basic concepts of the NFR methodology are explained and important definitions for terms like elementary quality attributes, non-functional requirements, or quality models are given. As quality models also play an important role in this NFR methodology, the role, typical elements, and the chosen representation for the quality models are introduced. Finally, a definition of what completeness means for the set of non-functional requirements in the context of this thesis is given.

Chapter 3 describes how to specify the non-functional requirements together with the functional requirements in an integrated way. It presents a summary of the state of the practice and the state of the art with regard to the existing elicitation and specification approaches. This also includes a summary on existing NFR frameworks and approaches and a clarification on the relationship between functional and non-functional requirements specification. This chapter further describes which challenges occur if NFRs are to be specified with functional requirements in an integrated manner and how to resolve this challenge, i.e., it determines the concrete locations for specifying the different types of NFRs.

Chapter 4 presents the elicitation algorithm used in the NFR methodology. It describes the difference between the NFR elicitation approach used in this NFR methodology and existing approaches. The elicitation algorithm is described, followed by a description of how the elicitation algorithm is operationalized in elicitation checklists that guide the NFR elicitation.

Chapter 5 describes the actual NFR elicitation and specification process, i.e., how the artifacts from Chapters 2, 3, and 4 are organized into a coherent process that enables effective and efficient elicitation of a complete set of non-functional requirements. After an overview of the overall NFR process, each process activity is defined in detail, presenting the process activities purpose, inputs, steps, outputs and an illustrating example.

Chapter 6 presents possibilities for systematically focusing the effort for the NFR elicitation and describes the existing tool support. The NFR methodology foresees several places where focusing can be used to

trade-off result quality and effort to be spent. Both, tool support and focusing shall increase the methodology's efficiency, making the NFR methodology more pragmatic and applicable to real-life projects.

Chapter 7 presents the validation of the NFR approach. It gives an overview of a series of eight real-life case studies that used the NFR methodology. After introducing the setup of the validation, we present an overview of the case studies contexts. The method was used in different domains in projects of different sizes. After that, the hypotheses from Section 1.2 are refined to a measurable level by using the GQM approach. Finally, we present and discuss the results from the case studies, the threats to validity of the results and questions that remain open for future empirical investigation.

Chapter 8 summarizes the results and the contribution. Furthermore, it discusses limitations as well as potential future work for the methodology, the empiricism, and the tool support.

2 Classifications, Definitions and Quality Models

This chapter lays the foundation for the NFR methodology. It starts in Section 2.1 with a description of the current state of the practice and the state of the art with regard to non-functional requirements definitions, classifications, and existing quality models such as ISO 9126 [Iso01] as a basis for non-functional requirements elicitation. Based on the state of the practice and the state of the art, in Section 2.2, a requirements taxonomy that classifies the different kinds of requirements is presented. Furthermore, a metamodel that relates quality characteristics, functional requirements, and non-functional requirements is introduced. This section also explains the basic concepts for the NFR methodology and defines important concepts like elementary quality attributes, non-functional requirements, and quality models. As quality models also play an important role in this NFR methodology, the role, typical elements, and the chosen representation for the quality models are introduced. Finally, a definition of what completeness means for the set of non-functional requirements is given.

2.1 State of the Practice and State of the Art

The state of the practice and the state of the art differ a lot for the specification and elicitation of NFRs (see Section 3.1). For the definition of the terms functional requirements and non-functional requirements (see Section 2.1.1) and for their usage of quality models (see Section 2.1.2), the state of the practice and the state of the art do not differ too much.

2.1.1 Existing Definitions for NFRs

In almost any presentation that addresses the topic of non-functional requirements, one of the first discussion items after the presentation is usually the discussion on whether non-functional requirements exist, what distinguishes them from functional requirements, and how exactly they are defined. This holds for presentations in academia as well as for presentations in industrial settings. Therefore, it is important to know what exactly is meant by the term "non-functional requirement".

State of the Practice

For the state of the practice, the term is usually implicitly defined by the usage of requirements specification templates such as [IEEE98a], [IEEE98b], or the Volere shell [RR99]. There, NFR are documented in

separate sections with names of typical quality attributes like performance, maintenance, or usability. As today Wikipedia also serves as a source of information for the state of the practice, this definition is also given here. The term non-functional requirement is defined there as "a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. This should be contrasted with functional requirements that define specific behavior or functions".

State of the Art

[Gli07] gives a comprehensive overview of NFR definitions in standards and academia. For the purpose of this thesis, a modified, i.e., extended version of this overview is given in Table 1.

The difference between functional and non-functional requirements is an aspect that needs to be understood in order to deal with non-functional requirements better. To start with the functional requirements, [Dav93] defines that behavioral requirements (synonym to functional requirements) specify the inputs (stimuli) to the system, the outputs (responses) from the system and the behavioral relationship between them. [Ebe98] uses one of the most popular definitions for functional and non-functional requirements when stating "A requirement that describes not what the software will do, but how the software will do it is called a nonfunctional requirement (NFR)." This definition states that the functional requirements describe the "what" and the non-functional requirements describe the "how". The nature of "how" only implicitly directs us to quality aspects. This "how" should not be mistaken for the how as used by [Yu97], who uses the words what, how, and why in order to define functional requirements, design, and context/rationales. There, he uses "How the system shall do it" to refer to design. To clarify this distinction better, sometimes "how well" is used to characterize the NFR. It becomes apparent that the usage of the terms "what" and "how" cannot be sufficient as definition. [LWE01] also gives an implicit definition, but already uses the word "characteristic" for describing the nature of NFRs: Functional requirements are "the things our software systems are supposed to do" and non-functional requirements are "the characteristics you intend your software to exhibit". Definitions like [Dav93], [IEEE90], [IEEE98a] name examples of such characteristics, like performance, portability or reliability.

Source	Definition
Antón [Ant97]	Describe the non-behavioral aspects of a system, capturing the properties and constraints under which a system must operate.
Burge and Brown [BB02]	Describe desirable overall properties that the system must have.
Davis [Dav93]	The required overall attributes of the system, including portability, reliability, efficiency, human engineering, testability, understandability, and modifiability.
Ebert [Ebe98]	A requirement that describes not what the software will do, but how the software will do it is called a nonfunctional requirement (NFR).
Franch [Fra98]	How the system behaves with respect to some observable attributes like performance, reusability, reliability, etc.
Franch and Carvallo [FC03]	…we define quality requirements as restrictions over the quality model.
IEEE 610.12 [IEEE90]	Term is not defined. The standard distinguishes design requirements, implementation requirements, interface requirements, performance requirements, and physical requirements.
IEEE 830-1998 [IEEE98a]	Term is not defined. The standard defines the categories functionality, external interfaces, performance, attributes (portability, security, etc.), and design constraints. Project requirements (such as schedule, cost, or development requirements) are explicitly excluded.
Glinz [Gli07]	A non-functional requirement is an attribute of or a constraint on a system.
Jacobson, Booch and Rumbaugh [JBR99]	A requirement that specifies system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility, and reliability. A requirement that specifies physical constraints on a functional requirement.
Kotonya and Sommerville [KS98]	Requirements which are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet.
Lawrence, Wiegers, Ebert [LWE01]	The characteristics you intend your software to exhibit.
Mylopoulos, Chung and Nixon [MCY+92]	... global requirements on its development or operational cost, performance, reliability, maintainability, portability, robustness, and the like. (...) There is not a formal definition or a complete list of nonfunctional requirements.
Paech and Kerkow [PK04a]	Any requirement describing the quality of the system.
Robertson and Robertson [RR99]	A property, or quality, that the product must have, such as an appearance, or a speed or accuracy property.
Wiegers [Wie03]	A description of a property or characteristic that a software system must exhibit or a constraint that it must respect, other than an observable system behavior.

Table 1: Existing definitions for the term non-functional requirements

Often, the term "quality requirement" is used as a synonym for a non-functional requirement. An example of this is the definition of [FC03], [Fra98], who - similar to the definition in [BBF+01] - argue for arranging the characteristics that serve as a basis for the NFRs in quality models.

They define "quality requirements as restrictions over the quality model"[1]. With quality models they refer to quality models such as ISO 9126 [ISO01] (see next section for details). A broader scope for NFRs is set by [Ant97] who characterizes NFRs like this: "describe the nonbehavioral aspects of a system, capturing the properties and constraints under which a system must operate." In addition to characteristics or properties, constraints are also subsumed as NFRs. [Gli07] shares this notion in his definition. Other approaches tend to emphasize the functional aspects and try to avoid the notion of NFR. [ZJ97], for example, state that "requirements are supposed to describe what the desired machine does, not how it does it." With this statement, [ZJ97] would exclude NFRs from the system requirements. This is emphasized in another section in their paper, where they state that "soft" requirements, like systems needing to be secure, reliable or easy to use, are not subject to their requirements refinement until they are precise enough to be formalized with their approach. The paper "NFR's: Fact or Fiction" [BB02] specifically deals with this topic. It defines NFRs as desirable overall properties that the system must have.

In addition to classifications of NFRs into different quality categories such as performance, usability, etc., some works also introduce orthogonal classifications. [Ebe98] classifies NFRs into customer-oriented NFRs and developer-oriented NFRs. This is a distinction that is relevant for our NFR methodology as the elicitation of NFRs will focus on different stakeholders. [Gli05] and [Gli07] give an overview of other existing classifications and an own, which Glinz calls a faceted classification (see Figure 4). According to his classification, non-functional requirements are requirements that are of the type "performance" or "specific quality constraint". An interesting classification is the satisfaction facet: Hard requirements are those that are either completely satisfied or not. Soft requirements can be gradually satisfied (similar to the satisfaction functions suggested by [LAG07]). In addition, the representation facet determines whether a requirement is specified in an operational, quantitative, qualitative, or declarative way. The role facet determines whether a requirement is prescriptive, i.e., a typical system requirement, or a normative or assumptive domain requirement. So we can say that NFRs can be classified with regard to several dimensions, including the quality attributes they are related to, their visibility (customer vs. developer), their measurability, and their relevance.

[1] An interesting observation is the trend of moving away from the "what" and "how" definition to a more quality model based definition. For example, Franch used in his early definitions [Fra98] the term "how", and in his definition in [FC03], he switched to the quality model based definition.

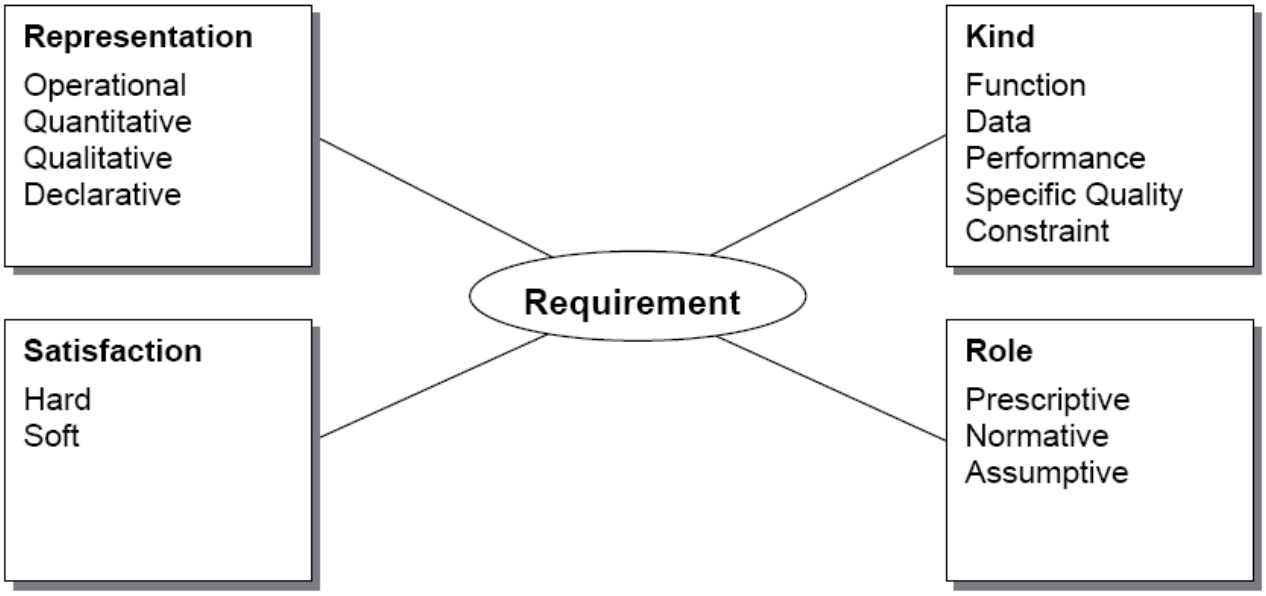


Figure 4: Faceted classification of requirements (figure taken from [Gli07])

Non-functional requirements are often addressed by goal-oriented approaches such as [MCY+92] or [Lam01a]. In this context, the distinction between functional and non-functional goals is of interest: [Lam01a] states that goals address functional and non-functional issues: "functional concerns associated with the services to be provided, and non-functional concerns associated with quality of service – such as safety, security, accuracy, performance, and so forth". [KKP90] gives a similar definition: "Functional goals underlie services that the system is expected to deliver whereas non-functional goals refer to expected system qualities such as security, safety, performance, usability, flexibility, customizability, interoperability, and so forth." One can see that the distinction on the goal level is almost identical to the one on the requirements level.

To summarize, almost all definitions agree that the functional requirements describe what the system is supposed to do, i.e., they describe the behavior or services of the system. Most definitions tend to describe non-functional requirements as information on or restrictions with regard to quality characteristics of the system. Some definitions make use of the term "quality model" to describe non-functional requirements. The next section will therefore summarize the state of the practice and the state of the art with regard to quality models. None of the existing definitions is precise enough for the NFR methodology in this thesis; therefore, a specific definition of the term non-functional requirements is given in Section 2.2.2.

2.1.2 Existing Quality Models

Generally, quality models are a list or hierarchy of quality attributes (also known as quality aspects or quality factors). According to [Mil00], who surveyed the most popular existing quality models, "The elements which describe the quality of a piece of software are usually referred to as quality factors, and collectively they are usually referred to as a software quality model".

To start with the state of the practice, the NFR categories in the templates of the IEEE 830 and 1362 standards [IEEE98a], [IEEE98b], the popular Volere shell [RR99], and the ISO 9126 standard [Iso01] name quality attributes and are typically used as implicit or explicit quality models for specifying NFRs.

Qualities in IEEE 1362

The IEEE 1362 standard, which is often used as a template for high-level requirements specifications (Concept of Operations Document), suggests specifying the following information as part of Section 3.3 "Description of the current system or situation" and also as requirements for the new system in Section 5.3 "Description of the proposed system":

- "Performance characteristics, such as speed, throughput, volume, frequency"
- "Quality attributes, such as: availability, correctness, efficiency, expandability, flexibility, interoperability, maintainability, portability, reliability, reusability, supportability, survivability, and usability"
- "Provisions for safety, security, privacy, integrity, and continuity of operations in emergencies".

Qualities in IEEE 830

The IEEE 830 standard which is often used as a template for more detailed requirements documents, suggests specifying requirements for the following quality attributes:

- Performance (Section 3.3)
- Reliability, availability, security, maintainability, and portability (all in Section 3.5).

Qualities in Volere

In the Volere shell, more specifically in the Volere requirements specification template, Edition 9, the following sections are listed in the chapter on non-functional requirements: look and feel requirements, usability requirements, performance requirements, operational requirements, maintainability and portability, security requirements, cultural and political requirements, legal requirements. Some of them are refined into more fine-grained sub-attributes.

Qualities in ISO 9126

The standard ISO 9126 is used in the state of the practice as well as in the state of the art. The ISO 9126 standard defines a quality model as "the set of characteristics and the relationships between them which

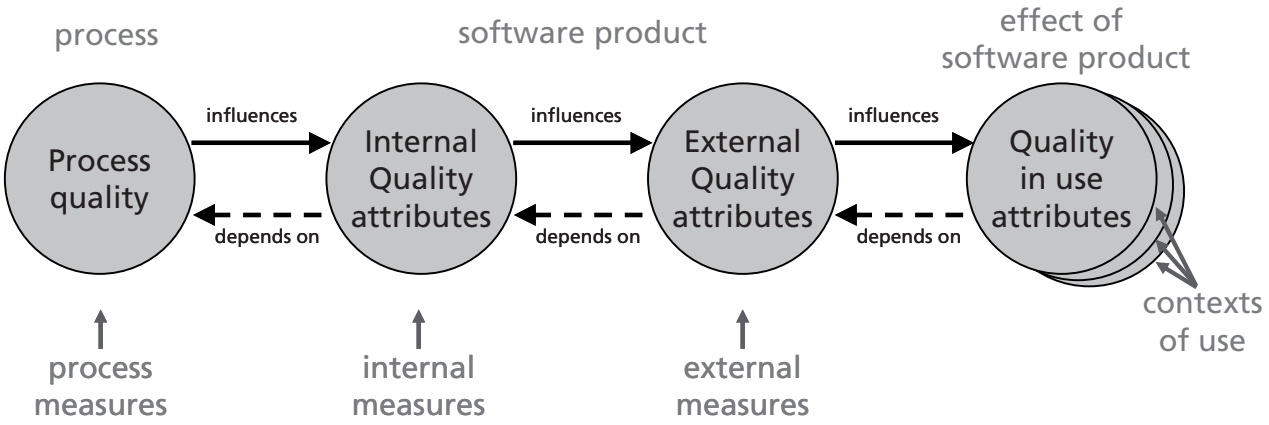


Figure 5: ISO 9126 view on qualities

provide the basis for specifying quality requirements and evaluating quality". An important distinction in the ISO 9126 standard is the differentiation of qualities into different levels of qualities (see Figure 5). There are internal, external and quality in use attributes. The quality in use attributes are quality attributes that can only be measured (and therefore also specified) in the context of their use. The quality attributes of this level are illustrated in Figure 7. The ISO standard defines these quality attributes as those perceived by the user. They are used to evaluate the effect the software product has on its environment. Most of the time, subjective metrics are used to measure these quality attributes. ISO 9126 also defines the internal and external quality attributes. It establishes a first level of quality attributes with six characteristics. Each is then refined into sub-characteristics. Strong similiarities exist to the categories of the aforementioned templates. Metrics that are intended for measuring the qualities are given in the remaining parts 2-4 of the ISO 9126 standard. Those metrics do not only assist in measuring product quality, but can also serve as help for expressing NFRs (more information on the usage of metrics to express NFRs can be found in Section 2.2.2). Therefore, the internal and external quality attributes are the typical quality attributes for NFR elicitation and specification (see also Figure 6). Sometimes, NFRs are also expressed for quality in use attributes, but typically, these are not the kinds of attributes used to specify NFRs, as they are too context-dependent. Work on using quality in use quality attributes for requirements engineering can be found for example in [DHK+07], [DKL+08].

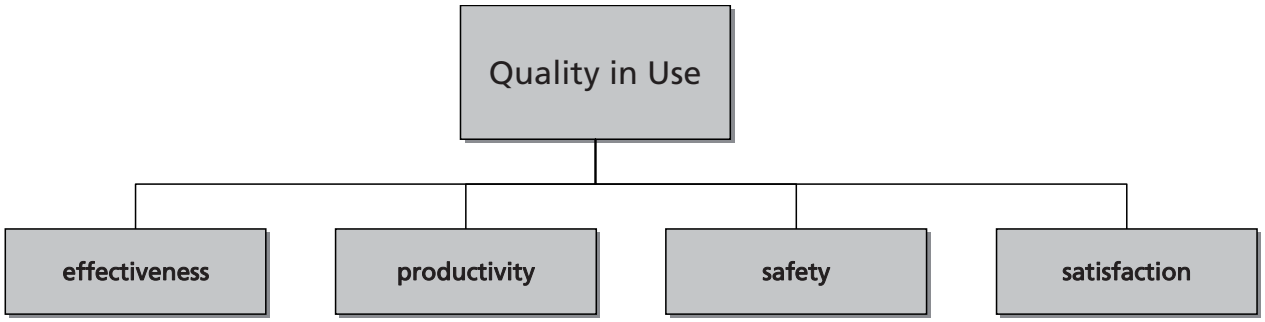


Figure 7: ISO 9126 quality model for quality in use

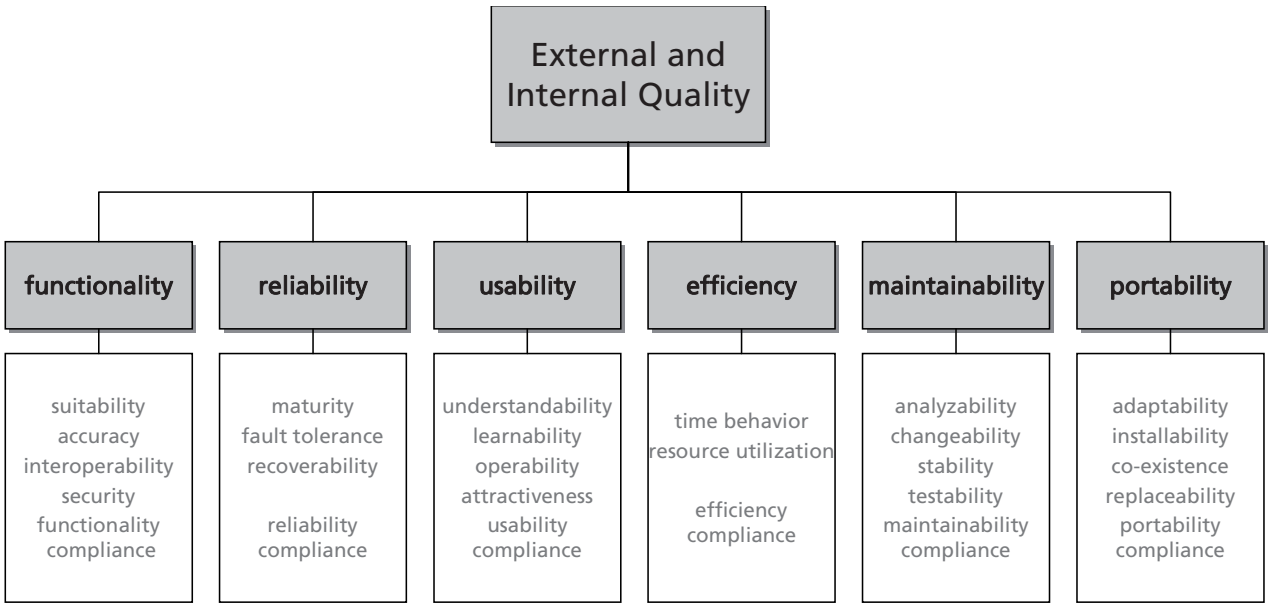


Figure 6: ISO 9126 quality model for external and internal attributes

Other Quality Models

In the state of the art, other quality models have also emerged. In 2000, Miller [Mil00] performed a survey on existing software quality models. These quality models include:

- Boehm from 1976 [BBL76],
- Cavano & McCall, which originated from studies done at USAF in 1978 [CM78],
- the FURPS+ model from Hewlett Packard from the mid-1980s [Gra92],
- the Garvin and Plsek's Dimension of Quality from the end of the 1980s [Gar88], [Pls87],
- the 1991 version of the ISO 9126 standard [ISO91],
- the SEI model from the mid 1990s [BKL+95].

[Mil00] comes to the conclusion that none of the quality models completely subsumes any of the others and that ISO was well aware of the models of Boehm, FURPS+, and Cavano and McCall. When choosing a model, he recommended the ISO9126 model as being among the most comprehensive and most frequently used one. Since then, few additional models and attributes have emerged. The 2001 version of ISO mainly added some sub-attributes for the first-level attributes usability and portability. [BLF02] in 2002 also summarize classifications that are composed of Boehm's model [BBK+78], McCall's model [McC94], and, additionally, classifications by Kirner and Davis [KD96], Mylopulous et. al. [MCY+92], and Yeh et. al. [YZC+84]. But the additional classifications do not yield significantly more quality attributes.

No Detailed General Purpose Quality Model

One conclusion of this is that there is no such thing as a detailed quality model that fits all purposes. If they need to be detailed, the quality models have specificities depending on the domain of usage. [FC03] also argues that there is no single general-purpose quality model; such a quality model always has to be tailored to a domain. They state that this domain-specific breakdown of quality attributes to a measurable level is usually not trivial. The NFR methodology in this thesis will therefore make use of experience-based quality models that can, but do not need to, be based on such quality models such as ISO or Boehm's model.

Other Sources for Quality Model Information

Another source for quality model information is experience on typical refinements of qualities. This can emerge due to approaches like [CNY+99] or [CL01a]. They use graphs to decompose high-level attributes into lower-level attributes. Mostly these decompositions are performed for specific products rather than for general purposes. Nevertheless, interesting refinement information can be extracted from these models. [CNY+99], for example, has a rich taxonomy of non-functional goals. The MOQARE approach [HP08] also uses this project-specific refinement to decompose quality information from high-level goals into design decisions called countermeasures. In [CL01a], [CL01c] it is stated that they store knowledge about typical decomposition of quality attributes and interdependencies in a tool called the OORNF tool. This project-specific refinement information could be abstracted to create experience-based quality models for certain domains.

Emerging New Quality Models

Currently, there are two efforts to create new, more comprehensive quality models. First, the ISO 9126 standard is going to be revised and incorporated into a new series of standards called "Software product Quality Requirements and Evaluation (SQuaRE)" in the ISO 25000 series [ISO05], [Boe08]. Second, the ongoing German project "Quamoco" [Qua09] aims at defining a general quality standard that makes intensive use of quality models. For this purpose, many quality models are being surveyed according to a classification scheme for quality models [KHM+09].

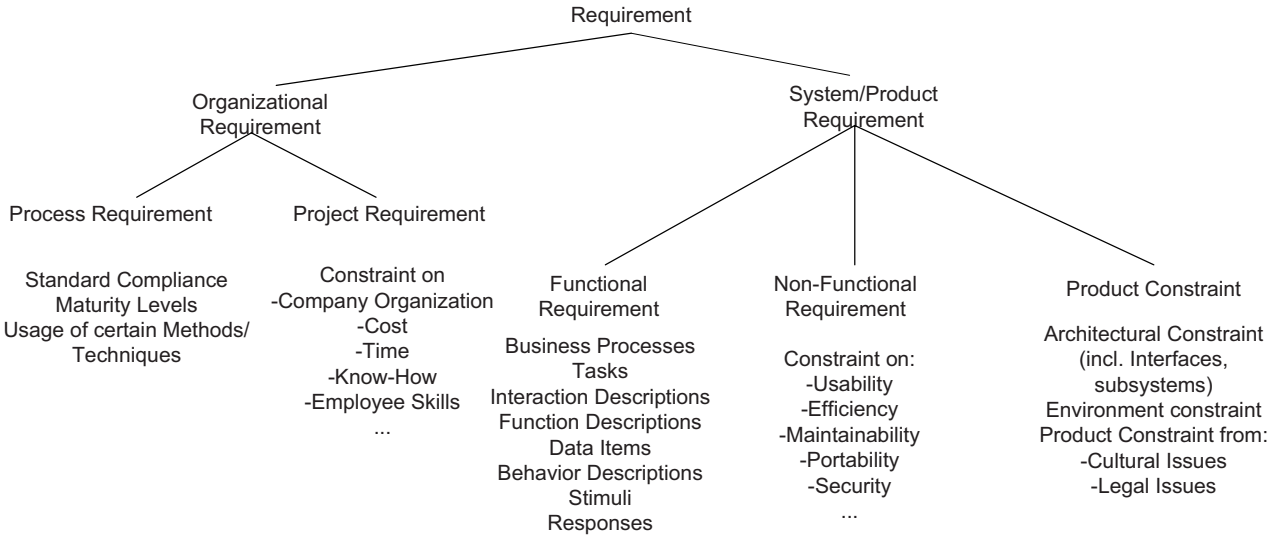


Figure 8: Requirements taxonomy

2.2 Taxonomy, Definitions, Metamodel & Quality Model

This section lays the foundation for the next chapters. Several terms used in this thesis are defined throughout this section. First, a requirements taxonomy is described that helps to distinguish the notion of NFRs used in this methodology from other approaches. Second, the metamodel for the NFR methodology is introduced. In this metamodel description, it becomes clear which functional elements are supported by this NFR methodology and which further elements are needed to formalize the NFR definition. Third, the quality model structure typically used for the NFR methodology based on ISO 9126 is presented. The definition for completeness for the set of NFRs is given in the fourth part.

2.2.1 Requirements Taxonomy

In order to define the notion of non-functional requirement for this thesis, we want to first introduce a requirements taxonomy. This taxonomy is based on the results of a joint workshop of German-speaking NFR experts from industry and academia during the course of the Working Group "Non-Functional Requirements" of the German Computer Society GI (Gesellschaft für Informatik), which was founded and is being led by the author. This basis [Doe09] is similar to the one found in [Rom09]. It was modified and the taxonomy shown in Figure 8 is the basis for the NFR methodology in this thesis.

The first differentiation is the separation of requirements into organizational requirements and system-related requirements (system requirements and product requirements are used synonymously). [Gli07] also

argues that project and process requirements are conceptually different from product requirements and should therefore be distinguished and separated at the root level. Organizational requirements address process and project issues like constraints on project cost or time, or the maturity level the development process has to fulfill. Requirements on specific development methods or techniques are also requirements in this category. Sometimes the term non-functional requirements comprises requirements from the category of organizational requirements. This is one of the reasons why the term non-functional requirements is sometimes regarded as fuzzy and is not well defined. For this thesis, the set of non-functional requirements is composed of requirements for the system to be built and not for the project or the process that enables the development of the system under development. As this term is frequently used during the course of this thesis, it will be defined now:

Definition:

*The **SUD (system under development)** is the system that should be developed from the given set of product requirements (functional and non-functional requirements and other constraints).*

As can be seen in Figure 8, the product requirements are distinguished into three groups:

- Functional requirements
- Non-functional requirements
- Product constraints

The taxonomy shows a list of typical types of functional requirements from different domains. Whereas business processes and interaction descriptions (often called operational scenarios) are typical for the information system domain with interactive systems, the terms stimuli, responses, and behavior descriptions are typical representatives of the embedded area. But even neutral terms like function descriptions and data items can be found in the category of functional requirements. After the introduction of the metamodel, Section 2.2.2.1 will present the key functional elements used in our NFR methodology.

The taxonomy already gives a first hint with regard to the non-functional requirements definition: One of the properties of non-functional requirements is their nature of restricting the solution space by constraining qualities. The term NFR will eventually be defined after introduction of the metamodel in Section 2.2.2.

The third SUD-related category of requirements are product constraints, which are usually known before the actual system development starts. These typically comprise constraints on the system architecture (like predetermined system components), constraints due to a certain culture the system is used in, constraints imposed by physical laws, constraints im-

posed by the operating environment, or legal constraints on the product to be developed (e.g., required functionality).

2.2.2 Metamodel

The taxonomy presented here only gives a rough classification for requirements. In order to better understand the nature, the differences, and the relationships between functional and non-functional requirements, a metamodel is introduced. The basic concepts in the metamodel will first be explained by using a simplified version of the metamodel; later, the full metamodel will be introduced.

Usage of the Metamodel

Besides explaining the basic concepts used in this thesis, the metamodel with its relationships lays the foundation for the definition of completeness of the set of NFRs. Furthermore, this formalization of the metamodel elements is used by the elicitation algorithm to systematically process the elements and to define precise end criteria for the elicitation process. The metamodel can therefore be seen as the first necessary formalization step to enable semi-automatic algorithmic processing of NFR elicitation.

Elements of the taxonomy presented in Figure 8 that are also represented in Figure 9 are:

- Product requirement
- Functional requirement
- Non-functional requirement
- Architectural constraint (as the subset of product constraints relevant for the NFR methodology)

In the metamodel, new elements are introduced.

Definition:
*A **Functional Conceptual Element** is an abstract concept that serves as a placeholder for different kinds of functional conceptual elements, like system functions or data items.*

Functional requirements describe the different functional conceptual elements. A more detailed characterization of functional conceptual elements will be introduced in Section 2.2.2.1.

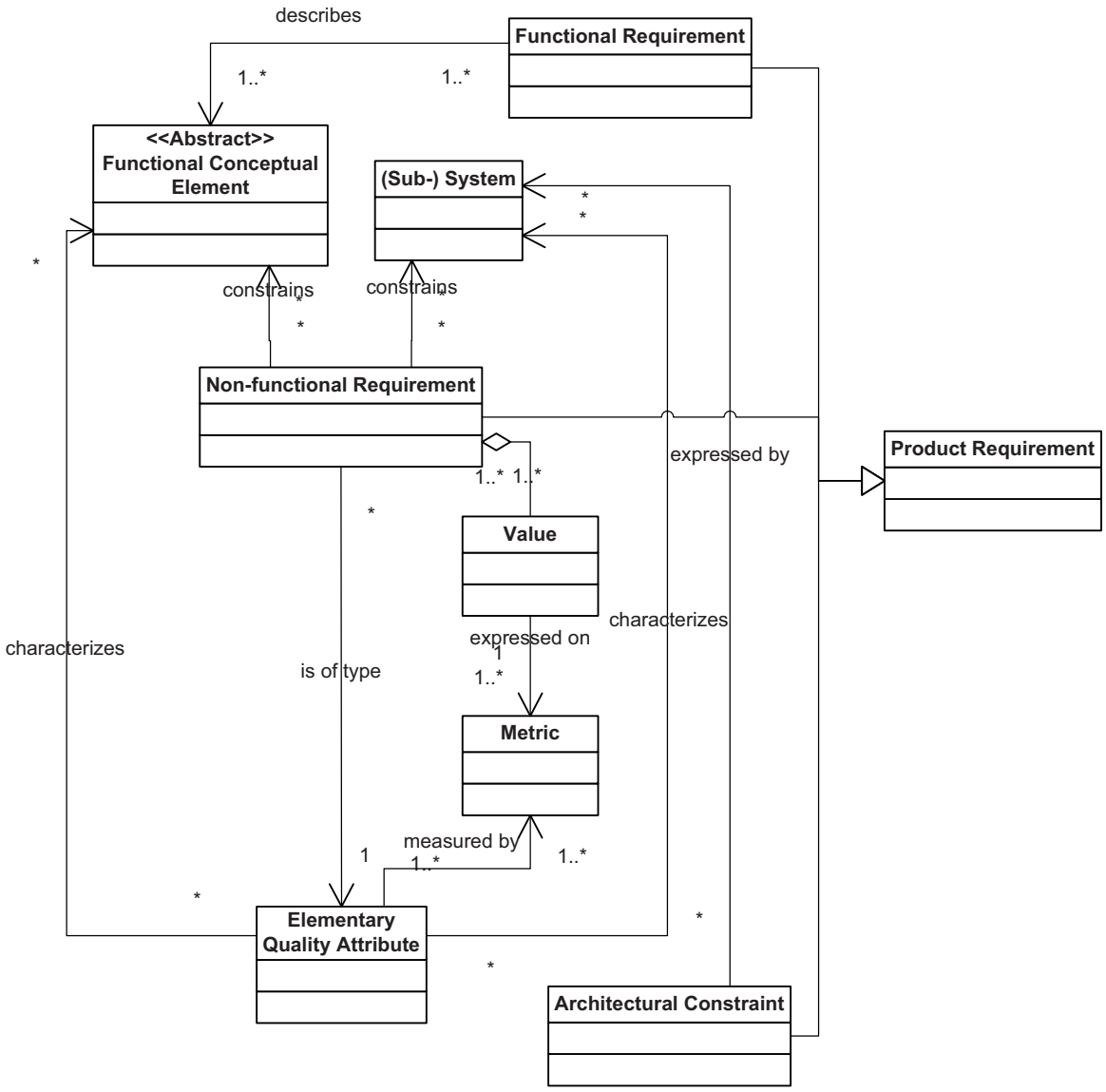


Figure 9: Simplified version of the metamodel

The concept *(sub-)system* refers to the system itself and its physical components that are predetermined by the architectural constraints, such as existing networks, servers, or predetermined end devices like mobile phones, etc. For ease of reading, we will use the term subsystem instead of the term (sub-)system. The term subsystem is therefore used as the set of objects containing the system and all its physical subsystems.

Elementary Quality Attribute

A key element in the metamodel for defining non-functional requirements is the introduction of the *Elementary Quality Attribute*. The elementary quality attributes are quality characteristics of the functional conceptual elements and subsystems. Therefore, they can be measured with the corresponding metric in an analytical usage, i.e., for quality assurance of the SUD. For NFR definition, we use them in a constructive way.

Definition:

*An **elementary quality attribute** is a measurable characteristic of one or more functional conceptual elements or subsystems that describes any other characteristic of this element but not its functionality (i.e., inputs, outputs, or input-output relationship). These other characteristics are often called quality characteristics.*

Typical elementary quality attributes are, for example, "response time", "network throughput", or "storage capacity". The concept of elementary quality attribute is intentionally defined by exclusion as in the definitions in the state of the art for NFRs.

An important part of the definition is the relationship to a functional conceptual element or subsystem. [MCY+92] already introduced a similar link between functional objects and quality attributes in their notion of non-functional goals, e.g, the term "Accuracy [attributes(employee)]" as a non-functional goal describes that the quality "accuracy" is related to the attributes of an employee, which is a data object.

Another important characteristic in the definition of elementary quality attribute is that one or more metrics can be attached to the quality attribute. A trivial example would be the metric "time in milliseconds" for the quality attribute "Response Time".

Definition:

*Set "**EQAS**"*

EQAS := the set of all elementary quality attributes for the development of a system.

More information on the hierarchy and relationships of elementary quality attributes can be obtained in Section 2.2.2.2. The completeness of the set of elementary quality attributes is one of the major success factors for obtaining a complete set of NFRs. Therefore, Section 2.2.4 provides some information on how to obtain a complete set of elementary quality attributes.

As already stated in Section 2.1.1, none of the existing state-of-the-art definitions is detailed enough to serve as a basis for the NFR methodology. Therefore, the following definition is given:

Definition:

*A **non-functional requirement (NFR)** constrains one or more functional conceptual elements or subsystems by determining values (or value domains) for one or more metrics of a specified elementary quality attribute that should be achieved by the functional conceptual element or subsystem. The non-functional requirement is of the type of the elementary quality attribute that characterizes the functional conceptual element or subsystem.*

This definition is similar to some definitions in the state of the art in the sense that a NFR describes a property or quality (elementary quality characteristic). As in the definition of elementary quality attribute, there are two important new aspects in this NFR definition.

Usage of Metrics

First, the usage of metrics ensures that the NFR is testable or measurable. For example, the NFR "The system function X shall be performed in 3 milliseconds." is of the elementary quality attribute type "response time" and constrains the functional conceptual element "system function X". The value "3 milliseconds" is determined based on the metric "time in milliseconds", which is a metric including measure associated with the elementary quality attribute "response time". But the whole area of metrics must be seen in a much more differentiated way: Many approaches such as [CNY+99], [LX99] and also [Gli07] state that there are soft or hard (rigid) goals and NFRs, respectively. In this NFR methodology, the rigidness of an NFR is expressed by the type of metric that is used to constrain a quality attribute. The level of measurability (objective vs. subjective) - whether it is an enumeration type (low, medium, high) or a numerical scale - makes the metric a flexible instrument for expressing soft and hard NFRs. [GS05] use so called worst-case settings to specify measurable NFRs. Another alternative for describing a metric would be the use of satisfaction functions as proposed by [LAG07]. In practice, it is also common to express values of metrics relative to existing or comparable systems, e.g., "the response time of the new system must be 10% below the response time of the existing system", rather than expressing absolute values like "10 ms". The work of [KOK04] uses this kind of information on changes to NFRs due to new product versions for identifying and specifying NFRs.

Relationship to Functional Element

Second, the NFR is directly linked to a functional conceptual element or subsystem. No consensus exists among the definitions in the state of the art in this respect. Whereas [MCY+92] supports this linkage, this link remains in contrast to the statement of [BB02], namely, that a main characteristic of NFRs is that "NFRs do not relate to a specific system component, instead they "cross-cut" software functionality." Whereas this statement might be true for some qualities, it is not for others. The introduction and differentiation of the functional conceptual elements is one key aspect to make an NFR elicitation possible based on functional conceptual elements and subsystem information. A linguistic analysis

[Eva95] stated that "non-functional requirements tend to be stated in terms of constraints on the results of tasks which are given as functional requirements (e.g., constraints on the speed or efficiency of a given task), a task-based functional requirements statement is a useful skeleton upon which to construct a complete requirements statement." And "It can be helpful to think of non-functional requirements as adverbially related to tasks or functional requirements: how fast, how efficiently, how safely, etc.". The next section will give details on typical functional conceptual elements. The set of NFRs is defined as follows.

Definition:

*Set "**NFRS**"*

NFRS := the set of all NFRs for the development of a system.

In order to express that an NFR is of the type of a certain elementary quality attribute, the relation isOfType is defined.

Definition:

*Relation "**isOfType**"*

$isOfType \subseteq NFRS \times EQAS$

$\forall n \in NFRS, \forall q \in EQAS : (n,q) \in isOfType$ *iff the NFR n is of the type of the elementary quality attribute q iff the NFR n uses a metric associated to q to express the value.*

Typical requirements documents are not limited to product requirements but also contain requirements with regard to the organization or development process. As explained in Section 2.2.1, we concentrate on the product-related requirements, namely the functional requirements, the non-functional requirements, and the architectural constraints.

2.2.2.1 Functional Elements of the Metamodel

Figure 9 showed a simplified version of the metamodel. Depending on the domain, different types of functional conceptual elements can exist. A typical inheritance of functional conceptual elements that can be used for the elicitation and specification of NFRs can be found in Figure 10.

As can be seen in Figure 10, the functional requirements describe functional conceptual elements. In the domain of interactive systems, these are typically:

- *Tasks*: We distinguish between two types of tasks:
  - *User tasks* are tasks that users have to perform with the system. They are supported by the system (e.g., "monitoring of certain machines"), but include some user involvement. The granularity of these tasks may vary a lot, ranging from view interaction sequences like in a typical use case to complex interactions as for a complete workflow.
  - *System tasks* (synonym for system functions or automated tasks) are tasks the system performs on its own. In contrast to user tasks, the user is not involved in system tasks.

Tasks can be refined into further tasks. Furthermore, user tasks can be refined into parts carried out by the user and system tasks (e.g., a user task "monitoring machine x" is refined into a set of system tasks such as "system displays alarm message if machine runs out of filling"). A task is typically described by one or more functional requirements (FRs). User tasks are typically described in the shape of Business Processes, Scenarios, or textual Use Case (UC) descriptions. System tasks are typically described by function descriptions. For the more complex system tasks, the algorithms are sometimes specified with activity diagrams.

- *Data items*: Especially in the area of information systems, data items

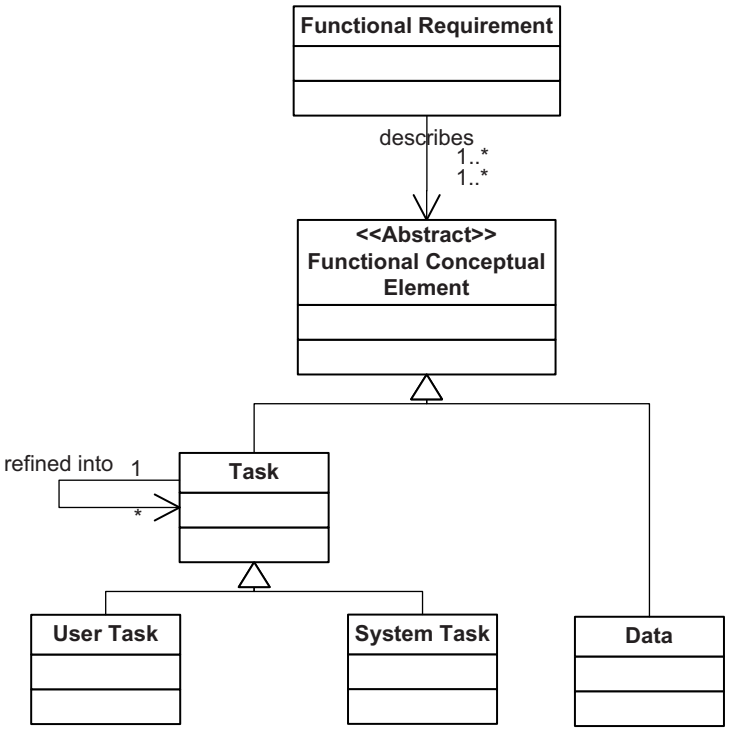


Figure 10: Functional elements of the metamodel

are entered into the system, manipulated in the system, and handed over to other systems or to the environment (reports, etc.).

That the functional part of the metamodel is based on these concepts is not surprising. In computer science, input data is transformed to output data by system functions. The interaction of the end-user with the system, encapsulated in user tasks, is especially important in the domain of information systems. These concepts are also the core concepts of state-of-the-art approaches such as the TORE approach [PK04b], [ADE+09].

During the course of this thesis, we will make use of several abbreviations. One important distinction is the difference between currently elicited items and the theoretical complete set (i.e., all items that should have been elicited). For the functional elements, the following definitions will be used:

Definition:
*Set "**UT_E**"*

UT_E := all user tasks that are specified in the current status of the requirements specification for the SUD

Within a user task, various steps are performed by humans and the system. Each step in a user task is either a human step or a system step.

Definition:
*Set "**UT_S**"*

Let $ut \in UT\_E$,
UT_S:= the set of all steps in the User Task ut

In case an $s \in UT\_S$ is performed by the system, this step of the user task is directly related to the corresponding system task st. In this case *UT_S* **uses** system function st. The set of elicited system tasks and elicited data items are defined as follows

Definition:
*Sets "**ST_E**", "**DI_E**"*

ST_E := all system tasks that are specified in the current status of the requirements specification for the SUD

DI_E:= all data items that are specified in the current status of the requirements specification for the SUD

The set of *UT_E*, *ST_E* and *DI_E* shape the set of elicited functional requirements that are specified in the current status of the requirements specification for the SUD.

Definition:
*Set "**FR_E**"*

$FR\_E := UT\_E \cup ST\_E \cup DI\_E$

In contrast to the elicited functional requirements, the theoretical complete set of functional requirements for the SUD is defined as:

Definition:
*Set "**FR**"*

FR := the set of all functional requirements that are needed to comprehensively describe the user tasks, system tasks, and data items. It is the set of functional requirements that should have been elicited in order to get a complete functional description for a system.

With the assumption that all elements in FR_E are correct requirements, $FR\_E \subseteq FR$ holds.

As user tasks are often refined into system tasks and this refinement is used in the elicitation algorithm, we want to define the "refines" relation:

Definition:
*Relation **"refines"***
$refines \subseteq ST\_E \times UT\_E$

$\forall ut \in UT\_E, \forall st \in ST\_E : (st, ut) \in refines \text{ iff } \exists y \in ut\_S \mid y \text{ uses } st$

This means that if and only if an elicited user task uses an elicited system task in one of the steps of the user task description, the elicited user task is refined by the corresponding elicited system task.

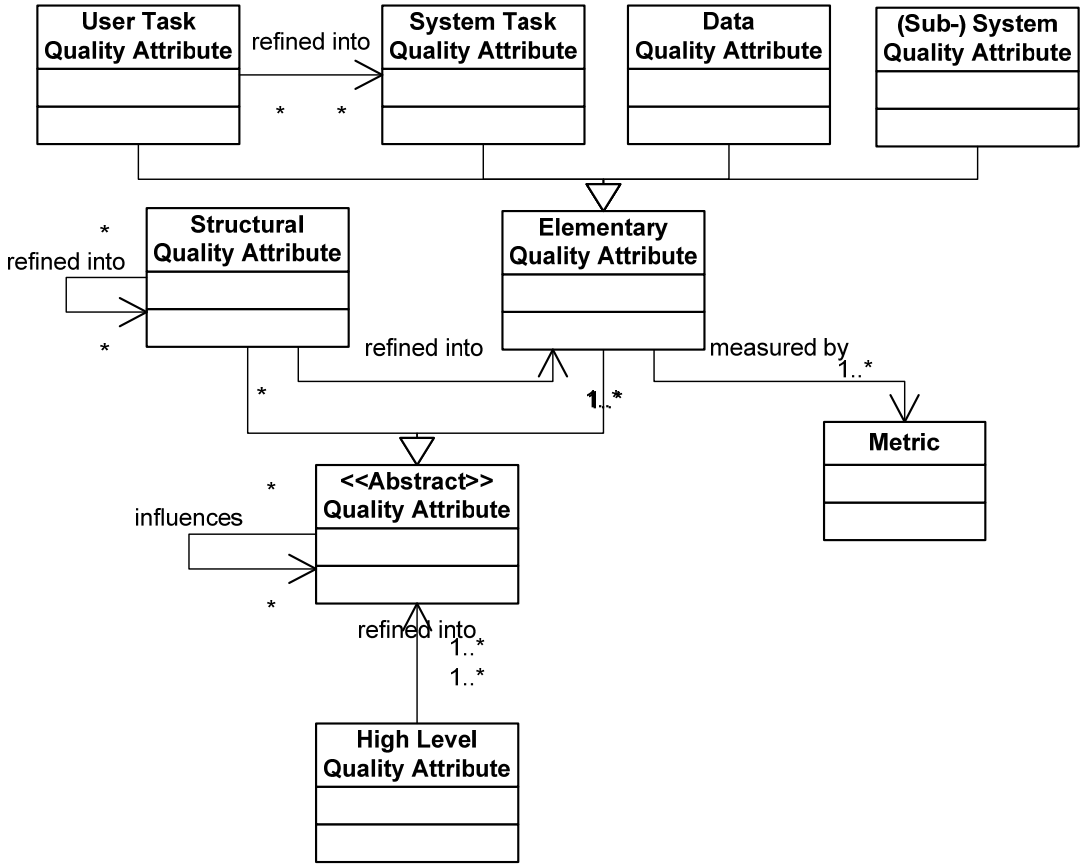


Figure 11: Quality related elements of the metamodel

2.2.2.2 Quality-related Elements of the Metamodel

The simplified metamodel in Figure 9 shows the concept elementary quality attribute. Figure 11 puts this concept into context. The elementary quality attributes are a specific type of quality attributes. The hierarchy of quality attributes starts with the high-level quality attributes (HLQA).

Definition:
*Set "**HLQA**"*

HLQA:= the set of relevant high-level quality attributes for the SUD

The elements of HLQA are typically not measurable. A typical example set is *HLQA = {Efficiency, Maintainability, Portability, Reliability, Security, Usability}*.

An element of the set HLQA is typically refined by further quality attributes.

Definition:

*A **quality attribute (QA)** is a quality (non-functional) characteristic of a functional conceptual component or subsystem. As in the definition of elementary QA, the term quality characteristic refers to any characteristic other than functionality.*

Quality attributes can influence other quality attributes in a positive or negative way. If the "workload", for instance, is higher, the "response time" might increase (example of a negative influence). This information will be relevant later for identifying NFR interdependencies. The concept quality attribute is abstract, as no instances of this concept exist. Instances can exist for the derived concepts "Structural QA" and "Elementary QA":

A quality attribute is of the following type:

- *Structural QAs* (depicted by the stereotype <<struct>>) such as "Time Behavior" are used to structure the elementary QAs into so-called quality models. Structural QAs do not have a metric attached and no non-functional requirement can be expressed directly on these QAs. Eventually, Structural QAs are refined to one or more elementary QAs.
- *Elementary QAs* were already defined at the beginning of Section 2.2.2. As can be seen in Figure 11, the elementary QAs can be of four different types:

– *System QAs* (depicted by the stereotype <<system>>), such as "Capacity", are measurable QAs that characterize the system or one of its subsystems (e.g., related to the database, secondary storage, or network). NFRs for these kinds of elementary QAs are typically elicited for the complete SUD or subsystems in the physical architecture of the SUD.

– *User Task QAs* (depicted by the stereotype <<utask>>), such as "Usage Time", are measurable QAs that characterize the tasks in which the system and the user are involved. NFRs for these kinds of elementary QAs are typically elicited for business processes or complete UCs, as these functional requirements exhibit the user/system interaction.

– *System Task QAs* (depicted by the stereotype <<stask>>), such as "Response Time", are measurable QAs that characterize system tasks, i.e., tasks that are carried out by the system, not including the user any more (e.g., calculation of results). NFRs for these kinds of elementary QAs are typically elicited for system functions, system features, or UC steps that are solely performed by the system, as these are the functional requirements that describe the pure system tasks.

– *Data QAs* (depicted by the stereotype <<data>>), such as "Precision of Data Storage", are measurable QAs that characterize data objects in the SUD. NFRs for these kinds of QAs are typically elicited for data items in the domain.

[LX99] gives a definition that a goal (which can be FR or NFR in content) has a view that is either actor-specific or system-specific. This notion can be found in this classification, as the actor-specific NFR goals are translated into User Task QAs (which relate to an actor) and the system specific NFR goals are translated into the System QAs.

In the following, we define the sets of the refined types of elementary quality attributes.

Definition:

*Sets "**QASYS**", "**QAUT**", "**QAST**", "**QADI**"*

QASYS:= the set of all elementary quality attributes for the SUD that characterize the system or one of its subsystems.

QAUT:= the set of all elementary quality attributes for the SUD that characterize the tasks in which the system and the user are involved.

QAST:= *the set of all elementary quality attributes for the SUD that characterize system tasks, i.e., tasks that are carried out by the system, not including the user any more.*

QADI:= the set of all elementary quality attributes for the SUD that characterize data objects in the SUD.

As can be seen in Figure 11, a User Task QA can be refined into System Task QAs. Therefore, we define the refines relationship for QAs as follows:

Definition:

*Relation **"refines"***

$refines \subseteq QAST \times QAUT$

$\forall utq \in QAUT, \forall stq \in QAST : (stq, utq) \in refines$ *iff the system task quality attribute refines the user task quality attribute iff the metrics associated to stq allow a refined expression of NFRs compared to the metrics associated to utq*

As can be seen in Section 2.2.3.2, QAs will be arranged in quality models in the shape of out-trees. Therefore, instead of using the relation refines to express the relationship, the terminology "the system task QA is a child system task QA of the user task QA" is also used.

In order to express that an elementary QA characterizes a functional conceptual element or subsystem, we define the relation characterizes as follows:

Definition:

Relation ***"characterizes"***

$characterizes \subseteq EQAS \times FR \cup SYS$

$\forall q \in EQAS, \forall f \in FR \cup SYS : (q, f) \in characterizes \text{ iff}$

$(f \in UT\_E \wedge q \in QAUT) \vee (f \in ST\_E \wedge q \in QAST) \vee$

$(f \in SYS \wedge q \in QASYS) \vee (f \in DI\_E \wedge q \in QADI)$

2.2.2.3 Complete Metamodel

Now that the functional and quality elements have been introduced, the complete metamodel is introduced to explain the remaining concepts and relationships.

Use of Rationale

A *rationale* justifies one or more product requirements. In the context of NFRs, a rationale states reasons for the NFR's existence, i.e., why it is important and cannot be neglected (e.g., "user will be unsatisfied if it takes more than 2 seconds to display alarm message"). [Bus09] states that often NFRs are unnecessary or extreme. Therefore, using a rationale can help to find out the justification for an NFR and help to eliminate such unnecessary or extreme NFRs.

The metamodel also depicts a "refine" relationship between NFRs: A non-functional requirement of the type "Usage Time" that states "The Use Case x must be executable within 1 minute" can be refined into several NFRs of the type "Response Time" that state that "Function X must have a response time of 1 second". This refinement relationship corresponds to the refinement relationship between tasks and between User Task QAs and System Task QAs, respectively.

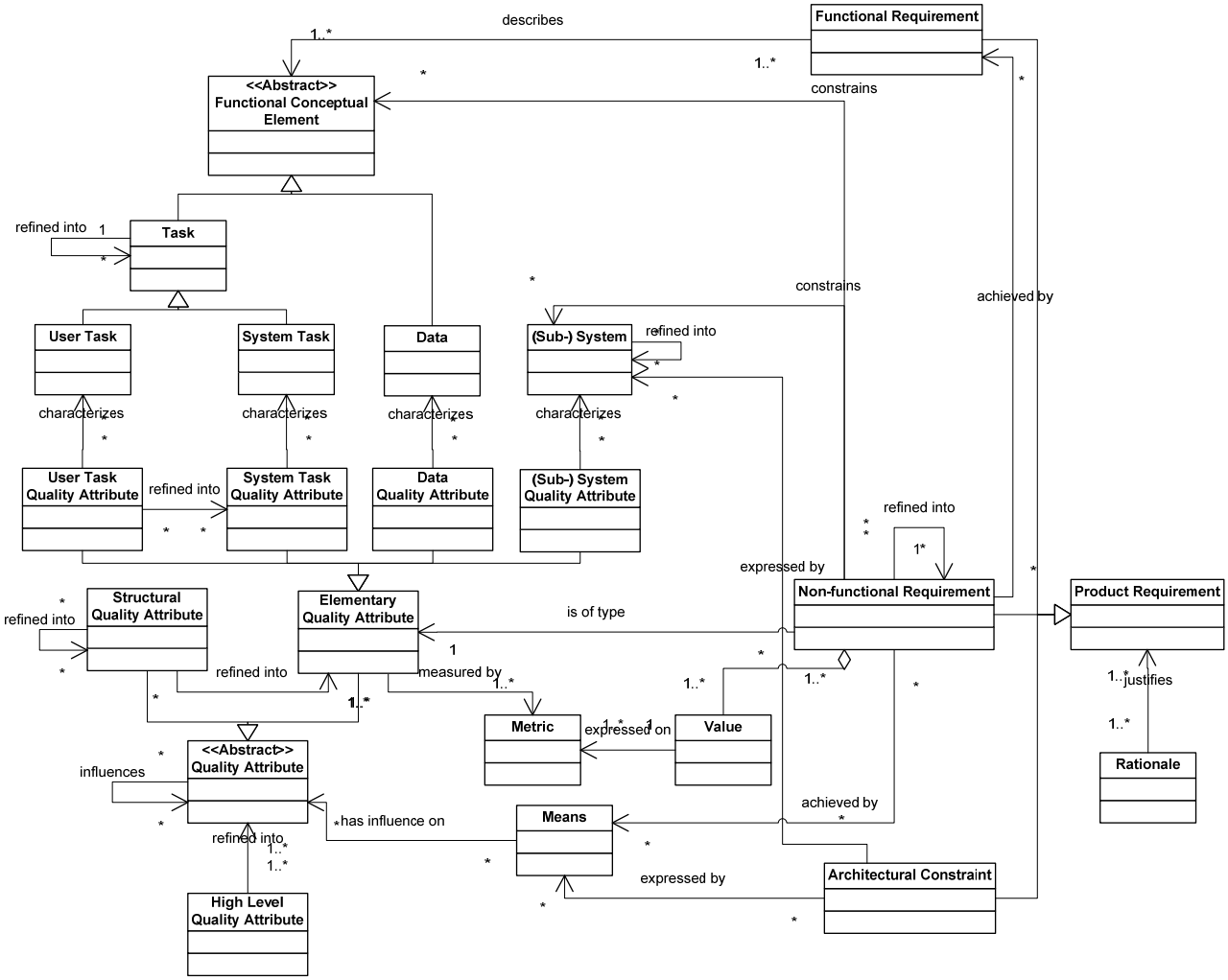


Figure 12: Complete metamodel for the NFR methodology

The metamodel further depicts an "achieved by" relationship between NFRs and FRs: This relationship explains a part of the reasons why some people state that there are no non-functional requirements. It is a common procedure that non-functional requirements are refined into functional requirements (or means, see following section). For instance, a security requirement that an authentication mechanism is needed can lead to (what we call "secondary") functional requirements demanding a log-in screen, an authentication algorithm, etc. But the origin of these secondary functional requirements is the NFR. Furthermore, there is an "achieved by" relationship between NFRs and means. An NFR can also be achieved by selecting a means to positively influence a quality attribute (see section on architectural constraints and means below).

The two "achieved by" relationships originating from the non-functional requirements can be compared to the relationships that [CNY+99] uses

to relate operationalizations (which are additional functional requirements and means in this metamodel) with softgoals.

Furthermore, the metamodel depicted in Figure 12 contains architectural elements. The architectural elements and relationships are included in the metamodel to show the relationships between non-functional requirements and architectural elements. This does not mean that these elements should be elicited in the requirements phase, but taken as given product requirements if they are predetermined. *Architectural constraints* constrain the SUD and can be of two types:

- expressing information on necessary *subsystems* (e.g., "the system shall have a database", "there will be a wireless network between the server and the mobile device"). The subsystems can again be refined into further subsystems. This information is relevant for the NFR elicitation, as System QAs characterize the subsystems that are already predetermined. Neglecting the subsystem information could therefore lead to missing NFRs of the type System QA.
- expressing the necessity to implement a *means*: A means describes an architectural decision that can be applied to the architecture to influence a certain QA and therefore achieve certain NFRs (e.g., "load balancing" is used to achieve a set of NFRs concerning the QA "workload distribution"). A means should have positive influence on the target quality attribute (but can also influence other QAs negatively as a side effect). The means are included in the metamodel for the following reasons:

– Means are not considered as requirements but either as architectural constraints or architectural decisions (if decided in the subsequent architecture development phase). Only if means are predetermined as architectural constraints is the constraint treated as a product requirement. This is also important, as means correspond to elements of other state-of-the-art approaches, e.g, operationalizations in [CNY+99].
– Means illustrate that the NFRs elicited can have an impact on the architecture phase in the sense that the selection of means should be based on the NFRs that are of the type of the QA that the means influence.

It is important to state that means are not always architectural constraints. A means can also be process- or project-related (e.g., the means "automatic test case generation" is used to improve the QA "reliability"). Means are most often selected and used by the developers (e.g., use of design patterns), but can also be visible for and demanded by the customer (e.g., documentation). An overview of some architectural and process-related means is given in [BH96]. More information on the relationship between the metamodel elements and architectural constraints can be found in [PvKD+03].

For the elicitation of NFRs, we take the information on subsystems as input.

Definition:
*Set "**SYS**"*

SYS:= the set of involved subsystems (databases, networks, server) for a SUD.

Please note that the system itself is also part of this set (see Section 2.2.2). The set SYS does not refer to software components, but high level physical components such as networks, database servers, etc. If the SUD is so small that there are no subsystems, the set SYS only contains one element: the system itself.

In order to complete the abbreviations for the types of requirements, we define the abbreviations for the sets NFR and NFR_E:

Definition:
*Set "**NFR**"*

NFR:=the set of all non-functional requirements that are requested by a stakeholder to comprehensively describe all related quality characteristics of user tasks, system tasks, data items and systems. It is the set of non-functional requirements that should have been elicited in order to get a complete non-functional description of SUD.

Finding all related quality characteristics, i.e., all relevant elementary QAs is a key to making the set of NFR complete. More information on how to obtain a complete set of elementary quality attributes can be found in Section 2.2.4.

Without the restriction "requested by a stakeholder", the set NFR would be the set of all non-functional requirements that attach a value for a metric for all elementary quality attributes to all functional conceptual elements they characterize. In practice, many of these statements are not needed, as a characterization of the functional conceptual element with a certain elementary quality attribute is not requested by any stakeholder. Therefore, the set NFR contains only those statements where a stakeholder request exists.

Next, we define the current state of elicited non-functional requirements (NFR_E).

Definition:
*Set "**NFR_E**"*

NFR_E:=the set of all non-functional requirements that are specified in the current status of the requirements specification for the SUD. It comprises the non-functional requirements wrt. data items, system functions, user tasks, and subsystems.

Assuming that all elements of NFR_E are correct, NFR_E $\subseteq$ NFR holds.

We refer to the product requirements by using the term REQ. The complete set of requirements relevant for NFR elicitation for the SUD is defined as:

Definition:
*Set "**REQ**"*

$REQ := FR \cup NFR \cup SYS$

2.2.3 Quality Models

Section 2.1.2 summarized the state of the art in quality models. On the one side, those quality models can serve as a good starting point for creating quality models. On the other side, they are not detailed enough for eliciting non-functional requirements. There is a need to handle experience-based, domain-specific quality models.

As can be seen in the metamodel (see Figure 12), the NFR methodology differentiates between NFRs and QAs. The various types of QAs described in Section 2.2.2.2 belong to the high-level quality attributes and should be arranged in quality models. For each of the elements of the set HLQA, a quality model is defined. The quality model gathers the quality attributes for an element of HLQA, puts them into a hierarchical order, and gives information on the relationship between the elements. For the remainder of this thesis, some definitions and abbreviations will be introduced:

Definition:
***QM_InScope$_{QA}$** is the project-specific quality model for the element QA $\in$HLQA. The quality model contains all quality attributes, metrics, and means and the information about the relationships between these objects in the model for the SUD.*

The project-specific set of all quality models in scope for SUD is defined as

Definition:

Set ***"QM_InScope"***

Let $QA \in HLQA$,

$QM\_InScope := \{QM\_InScope_{QA} \mid QA \text{ is in scope for the project}\}$

In some cases, we want to refer to the set of all quality attributes that are in a quality model for a given quality attribute.

Definition:

Set ***"$QA\_InScope_A$"***

$QA\_InScope_A$ *:=the set of all quality attributes in* $QM\_InScope_A$

To refer to the complete set of quality attributes in the scope for the SUD, we define QA_InScope.

Definition:

Set ***"QA_InScope"***

Let $A \in QM\_InScope$,

$$QA\_InScope := \bigcup_A QA\_InScope_A$$

2.2.3.1 The Role of Reference Quality Models

As depicted in Section 2.1.2, many different quality models exist. Experience has shown that there is no single, universal quality model that can be used in each and every context. Rather, one should start from so called reference quality models. The reference models can be obtained from different sources:

- Reference models that exist (or are created) in one's own organization
- Reference models from consultancy or research organizations or from state-of-the-art approaches, like [CL01a], [CNY+99], or [HP08]
- Reference models from the literature, like ISO 9126 [ISO01]

An adaptation of the reference quality models to the project context is usually necessary. Some approaches such as [RW07] also propose methods to create such reference quality models.

We first need to define some more terms that refer to the usage of reference models in contrast to the quality models that are specific for a concrete project.

The term QM_Ref_{QA} refers to the reference quality model for the quality attribute QA. In contrast to QM_InScope, this set comprises all quality attributes for the $QA \in HLQA$ that were gathered by experience (e.g., from former projects or projects of other companies), including those that are not relevant for the current SUD.

Definition:

QM_Ref_{QA} *is the reference quality model for the element $QA \in HLQA$. This quality model is not specific to the SUD but to a domain or company. The quality model contains all quality attributes, metrics, and means that are included in the model and the information about the relationships between these objects in the model.*

For future reference, we define the set of all reference quality models.

Definition:

Set **"QM_Ref"**

$QM\_Ref := \{QM\_Ref_{QA} \mid QA \in HLQA\}$

In some cases, we want to refer to all the quality attributes that are in a reference quality model for a given quality attribute.

Definition:

Set **"QA_Ref_A"**

$QA\_Ref_A :=$ *the set of all quality attributes in* QM_Ref_A

In contrast to QA_InScope, which depicts all quality attributes that are in scope for the SUD, we define QA_Ref as all quality attributes that exist in the reference models.

Definition:

Set **"QA_Ref"**

Let $A \in HLQA$,

$$QA\_Ref := \bigcup_A QA\_Ref_A$$

Typical Elements of HLQA

If one wants to start building quality models, the first question to answer is which elements the set HLQA provides. Generally, the NFR methodology works with any kind of set of qualities. Experience has shown that the widespread standard ISO 9126 can serve as a good basis (see also Section 2.1.2 for details). The ISO 9126 standard, as the most commonly used standard in this area, offers two quality models, one for the internal and external quality attributes and one for quality in use. As already argued in Section 2.1.2, the quality model of the internal/external quality attributes is suitable for NFR elicitation and specification. It suggests a set of six attributes, functionality being among this set. The attribute of security is hidden under the attribute functionality, and safety does not appear on this level (it is suggested in the quality in use model). The term functionality in the standard is quite misleading. Therefore, a typical starting set of elements for HLQA is the set of elements depicted in Figure 13.

In the following, we want to sketch the reasons why the typical set deviates from ISO 9126: First, we experienced that security is conceptually on the same level as the other QAs. Also, industrial partners view security as equally important as reliability, usability, etc. Therefore, we elevated security to the top level. Second, we removed the term "functionality" from the standard set HLQA as it leads to confusion. The target of this NFR methodology is to express NFRs as constraints on functional conceptual elements and subsystems. The functional capabilities of a product are expressed by its functional requirements. The other subcharacteristics of the ISO functionality characteristics (among them interoperability and accuracy) were moved to the quality models of the other elements of HLQA.

After having identified the set HLQA, the next relevant step for the NFR methodology is to determine the viewpoint (visibility) of the QAs in the quality models. ISO claims that the attributes are internal and external attributes. For the purpose of NFR elicitation from a customer point of view, the clear focus is on externally observable attributes. But there are also circumstances that lead us to focus on the internal attributes, for in-

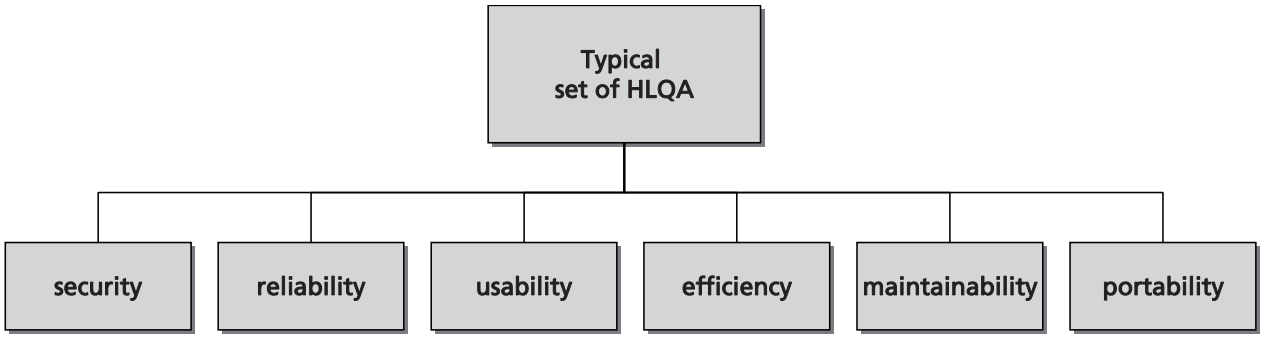


Figure 13: Typical set of HLQAs

stance if the stakeholder also has knowledge about internal attributes and wants to influence them (a situation that is often found in industry). Then, we do not restrict the elicitation of NFRs to external attributes.

2.2.3.2 An Adequate Representation for Quality Models

While there are common formats for specifying functional and non-functional requirements, we also need an adequate representation for the quality models. For example, it would be possible to use plain lists, trees, or graphs as representation forms. As one of the most important features of these quality models is the representation of the hierarchical relationship between the quality attributes, the tree representation is most adequate for this purpose. The plain list is not powerful enough to provide this overview. The graph representation would have the benefit that the non-hierarchical relationships (influences) between QAs could also be depicted and that a QA would not have to appear in two quality models, but the hierarchical character of the representation would be destroyed. Therefore, the quality models will be represented as trees (to be concrete: Out-Trees, i.e., trees with directed edges, going from the root to the children). The disadvantage of trees compared to graphs is compensated by using a separate matrix for capturing the non-hierarchical dependencies. Furthermore, quality attributes are replicated in case they must appear in two places in the quality model. As quality models are trees, the common tree-specific terminology can be used.

Figure 14 illustrates an example reference model for the quality attribute efficiency ($QM\_Ref_{Efficiency}$) as an out-tree. All QAs are arranged as unfilled rectangles. Means are depicted by filled rectangles and metrics are depicted as ovals. The element of HLQA for which this quality model is created is "Efficiency". This attribute is refined into the three structural child QAs: "Time Behavior", "Resource Utilization", and "Accuracy". The Structural QA "Time Behavior" is in turn refined into the User Task QA "Usage Time", which is refined into the System Task QA "Response Time". The metric for expressing the QAs "Usage time" and "Response Time" is seconds. NFRs can be expressed for all quality attributes that have an oval (metric) attached. The tree further describes that means like "Load Balancing" will positively influence the System QA "Workload Distribution". One can see that the box around the quality attribute "Network Topology" is in dashed lines. This should indicate that this quality attribute is usually not from a customer point of view but from a development point of view. Therefore, it is usually not the target of our NFR methodology. Example models that illustrate how company-specific models might look like can be found in Appendix A.

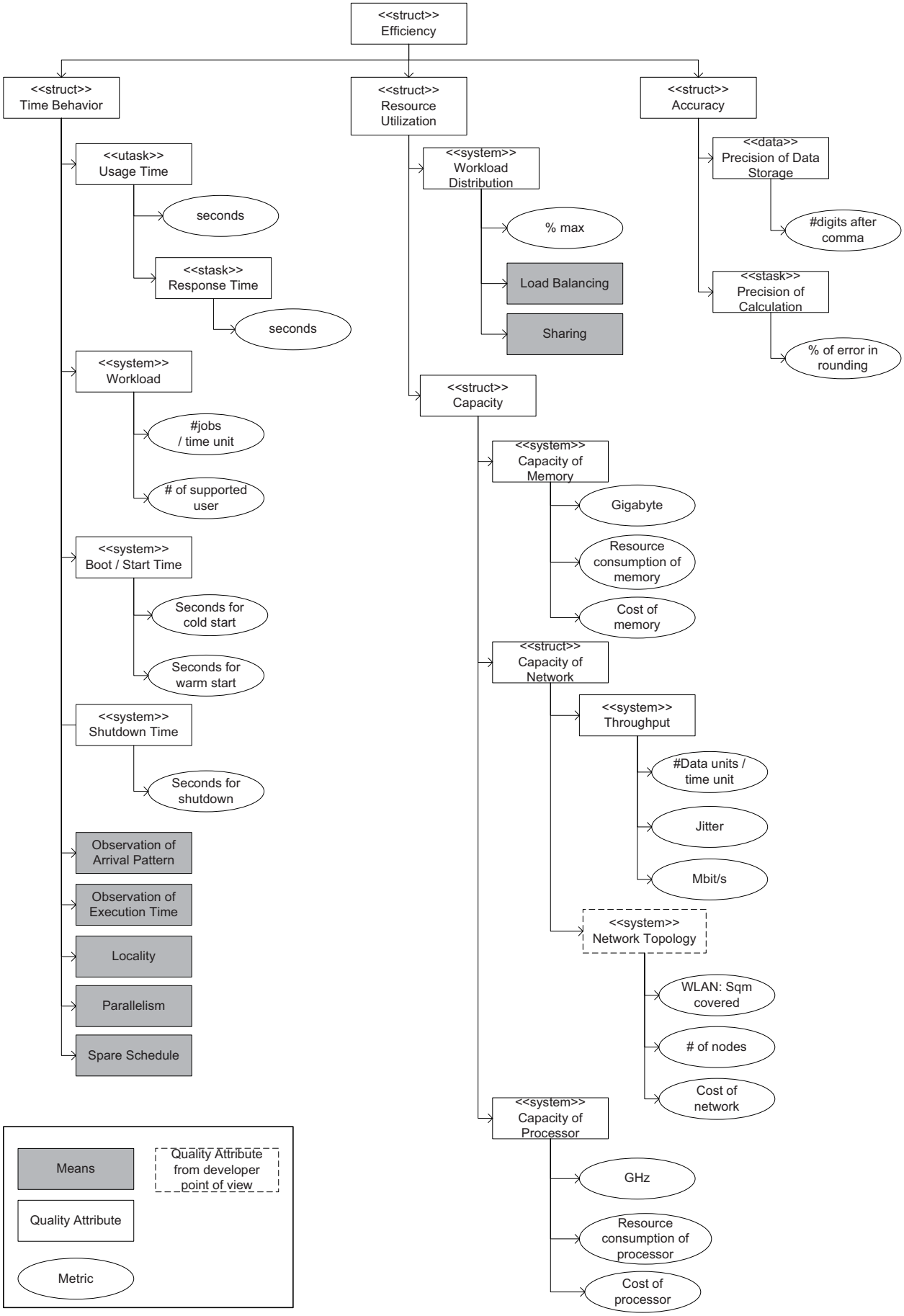


Figure 14: Example of $QM\_Ref_{Efficiency}$

2.2.3.3 Representing Non-Hierarchical Dependencies

As already described in the previous section, the out-tree representation of the quality model has the disadvantage compared to a graph representation that non-hierarchical dependencies within the quality model (so-called intra-quality model dependencies) cannot be incorporated.

The "refines" relationships between the quality attributes captured in the tree structure are used in a constructive way for eliciting the non-functional requiements. In contrast to this, the non-hierarchical "influence" dependencies are a crucial input for the analytical step of identifying conflicting NFRs after the elicitation of the NFRs. Therefore, we also need to represent the non-hierarchical dependencies. As non-hierarchical dependencies usually also exist between quality attributes of different quality models (we speak of inter-quality model dependencies), an additional representation of non-hierarchical dependencies is needed anyway.

Non-hierarchical dependencies are usually used to express that one QA might influence another QA, i.e., two QAs can be in conflict. This can also impact the NFRs that are of the type of the QAs. We will see later that NFRs also can be in conflict with each other. We define the relation conflict for elements of NFR.

Definition:

*Relation "**conflict**"*

$conflict \subseteq NFR\_E \times NFR\_E$

$\forall n_1, n_2 \in NFR\_E : (n_1, n_2) \in conflict$ *iff* n_1, n_2 *stay in conflict iff there is no solution for building a system that can satisfy* n_1 *and* n_2

For this definition, it does not matter whether the reason for the fact that there is no solution for building a system with the two conflicting NFRs is a technical reason (it is not possible with current technology) or an economical reason (it is too expensive to build such a solution).

As elements of NFR can be in conflict, we also have a similar relation on the level of elements of QA.

Definition:

*Relation "**influence**"*

$influence \subseteq QA\_Ref \cup QA\_InScope \times QA\_Ref \cup QA\_InScope$

$\forall q_1, q_2 \in QA\_Ref \cup QA\_InScope : (q_1, q_2) \in influence$
iff $\exists n_1, n_2 \in NFR \mid (n_1, n_2) \in conflict \wedge (n_1, q_1) \in isOfType \wedge (n_2, q_2) \in isOfType$

The interpretation of this definition is as follows: Two quality attributes influence each other if the potential exists that one can express corresponding NFRs that will stay in conflict.

In order to represent the non-hierarchical dependencies (influence-relation), the concept of dependency matrices is introduced. A dependency matrix represents relationships between $QA \in QA\_InScope$ or $QA \in QA\_Ref$. The following relationships that can be seen as a refinement of the influence relationship on QAs are represented in a QA dependency matrix:

1. A ↧ B: A influences B

2. A ↵ B: B influences A

3. A ↓↑ B: Bidirectional influence: A influences B and B influences A

4. A ⇉ B: A is identical with B (this is often used if the QA appears in more than one quality model)

Proportional dependencies (the higher A, the higher B) and inversely proportional dependencies (the higher A, the lower B) are distinguished by using background patterns in the dependency matrix:

- Proportional dependencies are marked with a dotted background pattern
- Inversely proportional dependencies are marked with a striped background pattern

In order to give a complete overview of the dependencies, hierarchical information that is already represented in the quality model is added by using the same symbols as the influence symbols, but with color coding (symbol is used with gray color):

- A ↧ B: A is part of B
- A ↵ B: B is part of A

If structural QAs are not the target of conflicting QA, they can be omitted from the matrix in order to make the matrix as small as possible.

	Usage Time	Response Time	Workload	Boot / Start Time	Shutdown Time	Workload Distribution	Capacity of Memory	Throughput	Capacity of Processor	Precision of Data Storage	Precision of Calculation
Usage Time		↵	↵	↵		↵		↵			
Response Time			↵			↵	↵	↵	↵		↵
Workload						↓↑	↓↑	↓↑	↓↑		
Boot / Start Time							↵		↵		
Shutdown Time							↵		↵		
Workload Distribution								↓↑			
Capacity of Memory										↵	↵
Throughput										↓↑	
Capacity of Processor											
Precision of Data Storage											↓↑
Precision of Calculation											

Figure 15: Example dependency matrix for $QM\_Ref_{Efficiency}$

Figure 15 is an example of such a dependency matrix that primarily shows the non-hierarchical dependencies for the $QM\_Ref_{Efficiency}$ shown in Figure 14. One can see, for example, that "Precision of Calculation" can influence "Response Time" (the higher the precision, the more time is needed), i.e., *(Precision of Calculation, Response Time)* $\in$ *influence*. This means that for two NFRs $n_1, n2 \in NFR\_E \mid (n_1,$ *Precision of Calculation)* $\in$ *isOfType* $\wedge (n_2,$ *Response Time)* $\in$ *isOfType*, it needs to be checked for n_1 and n_2 whether $(n_1, n_2) \in$ *conflict*. The quality attribute "Network Topology" was left out of the matrix for simplicity reasons (as a developer point of view quality attribute it is normally not considered in the NFR methodology).

Furthermore, we want to stress that for reasons of simplicity, we do not replicate information in the matrix: e.g., as workload influences usage time, we put a marker into the matrix in the column "Workload", row "Usage Time". We could have added the relationship in the opposite direction in row "Workload", column "Usage Time", but this would not add new information.

As already announced in the definition of the term quality model (see Section 2.2.3), we want to emphasize that the relationships information

depicted in the dependency matrices belongs to the quality model QM_QA for a $QA \in HLQA$.

2.2.4 Definition of Completeness

Now that the basic elements of the approach have been described, we want to explain what completeness means in the context of this NFR methodology. In Figure 12 we see that NFRs constrain elements of FR and SYS. For the definition of completeness, we introduce the abbreviation for the relation c.

Definition:

*Relation "**constrains**"*

$constrains \subseteq FR \cup SYS \times NFR$

$\forall f \in FR \cup SYS, \forall n \in NFR : (f, n) \in constrains$ *iff the NFR n constrains f iff the NFR n limits the possible solution space for fulfilling f*

To make the goal of this thesis clearer, the term completeness is defined.

Definition:

***Completeness** is defined as follows:*

***NFR_E is complete** iff NFR_E = NFR iff*

$$(\forall f \in UT\_E, \forall\ q \in QA\_InScope \cap QAUT:$$
$$(\exists n \in NFR\_E : (n,q) \in isOfType \wedge (f,n) \in constrains)$$
$$\vee(\neg\exists n \in NFR : (n,q) \in isOfType \wedge (f,n) \in constrains)) \ \wedge$$
$$(\forall st \in ST\_E, \forall\ q \in QA\_InScope \cap QAST:$$
$$(\exists n \in NFR\_E : (n,q) \in isOfType \wedge (st,n) \in constrains)$$
$$\vee(\neg\exists n \in NFR : (n,q) \in isOfType \wedge (st,n) \in constrains)) \ \wedge$$
$$(\forall d \in DI\_E, \forall\ q \in QA\_InScope \cap QADI:$$
$$(\exists n \in NFR\_E : (n,q) \in isOfType \wedge (d,n) \in constrains)$$
$$\vee(\neg\exists n \in NFR : (n,q) \in isOfType \wedge (d,n) \in constrains)) \ \wedge$$
$$(\forall s \in SYS, \forall\ q \in QA\_InScope \cap QASYS:$$
$$(\exists n \in NFR\_E : (n,q) \in isOfType \wedge (s,n) \in constrains)$$
$$\vee(\neg\exists n \in NFR : (n,q) \in isOfType \wedge (s,n) \in constrains))$$

In other words: The set NFR_E is complete if there is a non-functional requirement expressed for each pair of elicited functional conceptual element or subsystem and corresponding elementary quality attribute (e.g., in case of user tasks (UT_E), an elicited user task f and a User Task QA q), or there is no stakeholder request for it (which means there is no $n \in NFR$).

In the example in Figure 16, the one user task UT needs to be compared with the one User Task QA "Usage Time". The three system tasks marked with ST need to be compared with the one system task QA "Response Time" and the two subsystems SYS1 and SYS2 need to be compared with the System QA "Capacity". If there is no NFR expressed in NFR_E for such a pair (see filled circle in FR_E in Figure 16), then the set NFR_E is either incomplete or there is no NFR applicable (no stakeholder requests this NFR). In the situation in Figure 16, no stakeholder requests the NFR. If there is an NFR $n \in$ NFR that is related to a functional conceptual element or subsystem but not included in NFR_E (see filled square in Figure 16), then the set NFR_E is incomplete.

Assumptions

The set NFR_E is complete with this definition based on three assumptions:

1. The set of functional conceptual elements and subsystems is complete, i.e., all user tasks, system functions, data items, and subsystems were input to the NFR methodology.

2. The set QA_InScope is complete, i.e., all relevant quality aspects were determined in the quality models.

3. The stakeholder does not intentionally hide NFRs when asked for NFRs in the elicitation process. This might happen for political reasons or due to group pressure in a workshop.

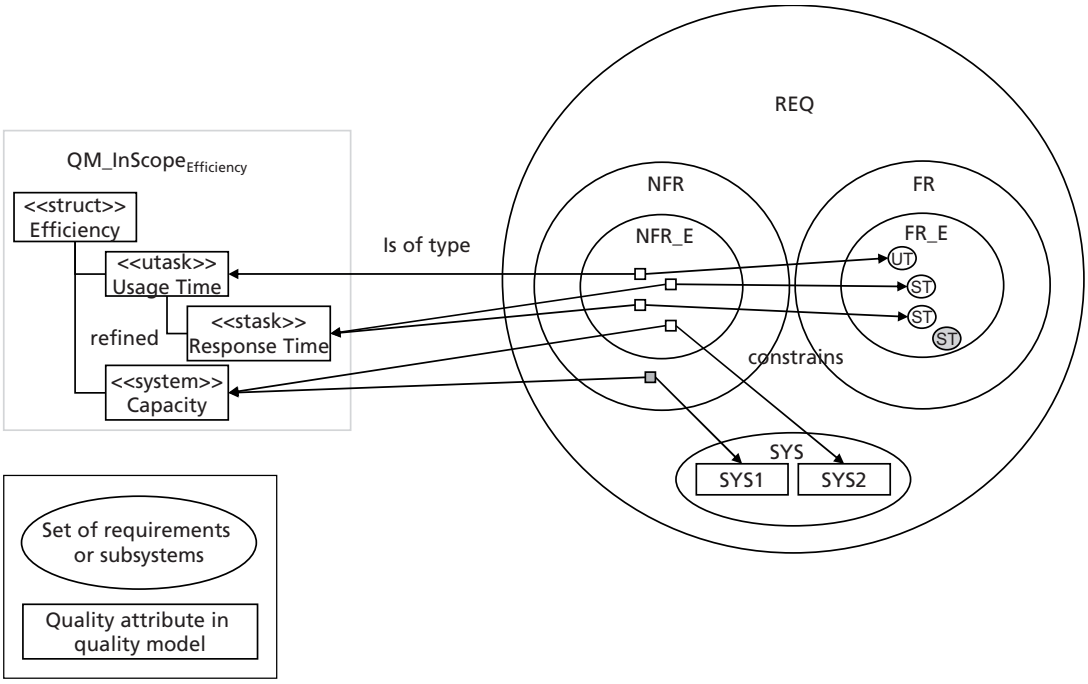


Figure 16: Example illustration for completeness definition

The second point is the most difficult one to assure. As each NFR constrains an element of $FR \cup SYS$ by expressing a value on the metrics of a QA, the set of QAs for the project QA_InScope plays a major role for making the set of NFR_E complete. An NFR can only be elicited if the corresponding QA is an element of QA_InScope. Therefore, getting a complete set of QA is a major objective of the NFR methodology and a kind of prerequisite for getting a complete set NFR_E. A major contributor to getting the completeness of the set QA_InScope is the existence of the reference quality models QM_Ref. They capture the existing know-how on the elements of HLQA, which is based on the literature or on knowledge from standard as well as from previous projects. So if a NFR was missed in a project because the corresponding QA was not in the model, this QA will usually be incorporated into QM_Ref so that this will not happen again in the future (for this learning from project experience, please refer to Section 5.8).

Limitation of Completeness Definition

The definition of completeness used in this thesis is limited in the sense that only those NFRs are addressed that originate from a relationship between a quality aspect and a functional conceptual element and subsystem, respectively. One could ask whether NFRs can exist that do not fulfill these criteria. Till today, no such NFRs have occurred in our work. But an absence of such NFRs is not proven. To prove such an absence, one would need a more comprehensive theory on which concepts NFRs are grounded. But none of the existing state-of-the-art approaches provides a more comprehensive set of concepts for grounding NFR completeness than the metamodel used in this thesis.

3 Specification of FRs and NFRs

This chapter will describe how to specify non-functional requirements and functional requirements in an integrated way. This structural information about documentation locations shall

1. enable the role in the software development process that elicits the NFRs to store these NFRs effectively, i.e., he or she stores each NFR in the correct location of the document.

2. enable the roles in the software development process that are interested in the NFR information to effectively and efficiently access this information. Effectively means that a person searching for NFRs finds all relevant NFRs, whereas efficiently refers to the time needed to find all relevant NFRs.

Chapter 4 will then describe how the elicitation is performed in a structured process. This chapter begins with a summary of the state of the practice and the state of the art with regard to the existing elicitation and specification approaches (see Section 3.1). This also includes a summary of existing NFR frameworks and approaches and clarifies the relationship between functional and non-functional requirements from an elicitation and documentation point of view. Section 3.2 describes the challenge in specifying non-functional requirements and functional requirements in an integrated manner and how to resolve this challenge, i.e., it determines the concrete locations for specifying the different types of NFRs.

3.1 State of the Practice and State of the Art

This section describes the state of the practice and the state of the art with regard to specifying non-functional requirements.

3.1.1 State of the Practice in Specifying FRs and NFRs

Usage of Natural Language

Currently, most industrial companies still use natural language to capture their requirements. Typically, they use corporate requirements specification templates based on standards like [IEEE98a] or [IEEE98b], or the popular Volere Shell [RR99]. As the Volere requirements specification template is available free of charge, it found widespread use. As an example of such a template, we illustrate the typical chapters of the Volere requirements specification in Figure 17. Volere, like the two IEEE stand-

PROJECT DRIVERS
1. The Purpose of the Product
2. Client, Customer and other Stakeholders
3. Users of the Product

PROJECT CONSTRAINTS
4. Mandated Constraints
5. Naming Conventions and Definitions
6. Relevant Facts and Assumptions

FUNCTIONAL REQUIREMENTS
7. The Scope of the Work
8. The Scope of the Product
9. Functional and Data Requirements

NON-FUNCTIONAL REQUIREMENTS
10. Look and Feel Requirements
11. Usability Requirements
12. Performance Requirements
13. Operational Requirements
14. Maintainability and Portability Requirements
15. Security Requirements
16. Cultural and Political Requirements
17. Legal Requirements

PROJECT ISSUES
18. Open Issues
19. Off-the-Shelf Solutions
20. New Problems
21. Tasks
22. Cutover
23. Risks
24. Costs
25. User Documentation and Training
26. Waiting Room
27. Ideas for Solutions

Figure 17: Recommended sections of the Volere specification template

ards, completely separates the NFR information from the functional requirements.

All standards give no concrete advice on notations to be used for specifying the information in the sections. Most companies use plain natural language. Some use sentence patterns like the ones described in [Rup07]. A very common technique is the usage of textual UCs as also proposed by the Unified Process [JBR99]. Often, templates similar to [Coc00] are used. Figure 24 shows a partially filled typical UC template. According to [Gli07], the RUP proposes to document NFRs as close to UCs as possible, and only to put the general NFRs into separate chapters.

Most often, the requirements are stored in text documents or spreadsheet documents. Sometimes Wiki-based solutions [URW+08] or database-oriented solutions are used to support both, the requirements specification and management. Typical requirements databases found in industry are IBM Telelogic Doors, Borland Caliber RM, and QA Systems IRQA. A comprehensive list of available tools can be found in [INC09].

One approach that found its way into industrial practice at some companies and that has a special focus on NFRs is the Planning Language (Planguage) approach of Kai and Tom Gilb [Gil05], [Gil07]. Some case studies like [Jac99] show that this approach can be beneficial for the specification of measurable quality requirements, i.e., measurable NFRs. Planguage differentiates between different requirement types, among them performance requirements. The term performance is quite misleading, as in Planguage it subsumes all quality characteristics, such as Usability, Reliability, Maintainability, and others. The approach suggests to start from quite abstract statements (called complex quality requirements). They are similar to the concept of QAs in our NFR methodology. These complex requirements shall be decomposed to a measurable level. Various tags are used for the specification of the measurable quality requirements. Besides the name of the requirement, the Tag "Scale" determines the quantification scale that shall be used for the requirement. The tag "Meter" describes the process that shall be used to measure how well the requirement is fulfilled. The tags "Past" and "Goal" determine the current and the intended value for the requirement. [Gil07] also proposes to write down quality requirements together with functional requirements, not in separate chapters.

Usage of Modeling Notations

In industrial practice, one rarely finds requirements specifications modeled extensively with UML diagrams using Case Tools. In the area of large business information systems, workflow modeling with EPCs [Sch99], UML activity diagrams [RJB99], and data modeling with tables, UML class diagrams, or entity-relationship diagrams is widespread. Sometimes UML state-charts [RJB99] are used to model state transitions of important data objects. Figure 24 shows an example using EPCs and UML class diagrams.

3.1.2 State of the Art in Specifying FRs and NFRs

As this thesis aims at capturing a complete set of non-functional requirements, the state of the art in specifying NFRs is relevant. Furthermore, the state of the art with regard to functional requirements is summarized if it is relevant for the NFR elicitation or specification. The goal of this section is not to provide a complete state of the art for all possible functional requirements approaches. Rather than that, Section 3.2 will introduce the TORE approach as a state-of-the-art and state-of-the-practice compliant approach to modeling functional requirements. The NFR methodology is built on top of this functional approach.

The intention of the NFR methodology developed in this thesis is to be usable for as many HLQA as possible, not being specific for a single quality attribute. With regard to the state of the art in non-functional requirements approaches, we therefore focus on frameworks that are applicable to more than just one HLQA. These are typically the goal-oriented approaches, such as [CNY+99], the most popular approach among these. Furthermore, we also describe approaches that originated from one specific HLQA, but broadened their scope to others. We therefore introduce the MOQARE approach [HP08] which uses concepts from security requirements elicitation, i.e., the concept of misuse cases, and broadened its scope to basically all quality attributes. We also introduce in more detail the UMD [BDA04] approach which originated from the dependability community and can be used for multiple quality attributes as well.

[Fra98] and [MCY+92] divide approaches for dealing with NFRs into product- and process-oriented approaches. According to [Fra98], process-oriented approaches "use non-functional information to guide the development of software systems". Product-oriented approaches deal with products from an evaluation point of view and focus on checking the non-functional characteristic. He also argues that a combination of both approaches is needed to treat NFRs successfully.

Goal-based NFR Approaches

Several approaches that deal with NFRs are based on goal modeling. According to [EYM06], a goal model basically consists of the following elements:

- A set of nodes representing hierarchically decomposed goals
- A set of edges representing relationships among goals
- A set of actors that are owners of goals.

A good overview of goal modeling is given by [Lam01a]. According to [Lam01a] "a goal is an objective the system under consideration should achieve". Typical goal-modeling approaches are the ones by Yue [Yue87], Robinson [Rob89], Berzins and Luqi [BL91], Darimont, Fickas and Lamsweerde [DFvL91], Jarke and Pohl [JP93], Mylopulous et al. [MCY+92], and Zave [Zav97]. According to [PK04a], the most popular and comprehensive approach with regard to NFR specification is the NFR Framework developed by Chung, Mylopulous et al. [MCY+92], [CNY+99]. The NFR Framework treats NFRs as goals. High-level goals are decomposed into trees of sub-goals and more detailed sub-attributes. In those trees, also called softgoal interdependency graphs (SIGs), interdependencies between goals are modeled as well. Figure 18 gives an example of a softgoal interdependency graph as a goal model for non-functional requirements. Goal modeling is also supported by the GRL (Goal-oriented requirements language) [GRL02a], [GRL02b], which is based on the i* framework [Yu93], [Yu97] and is intended to describe NFRs. It uses the three main concepts intentional elements (goal, task,

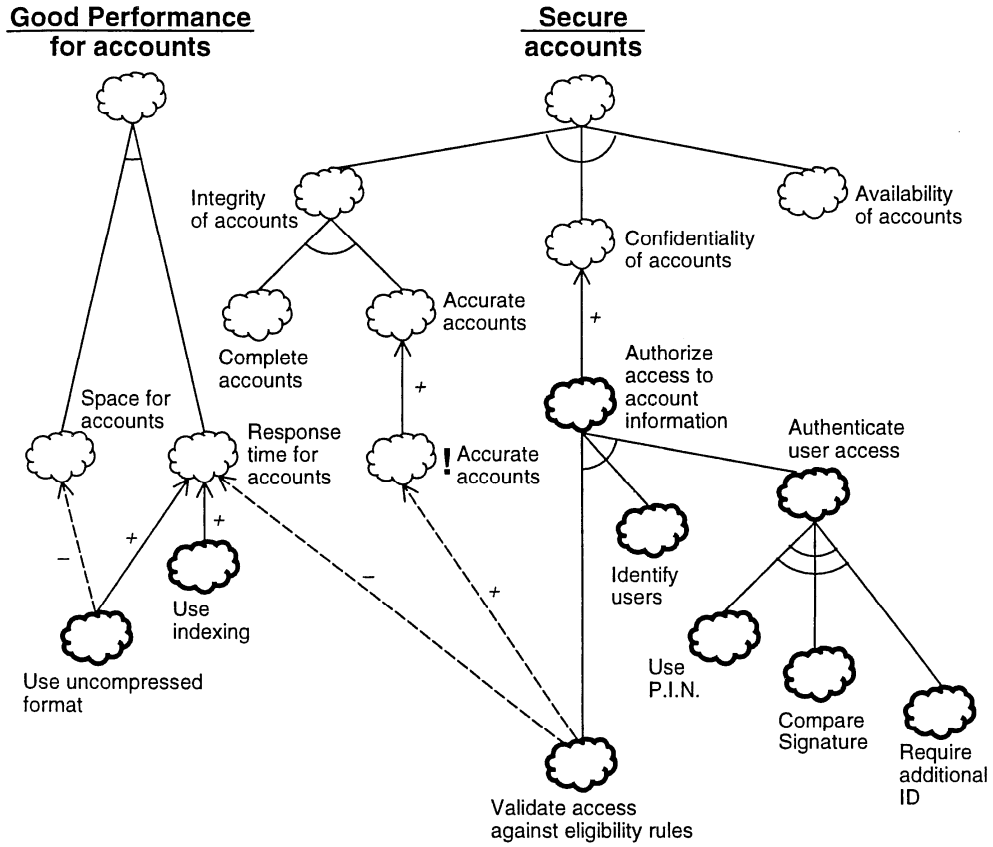


Figure 18: Example of a SIG as used in the NFR Framework (figure taken from [CNY+99])

softgoal, belief, resource), intentional relationships (means-ends, decomposition, contribution, correlation, dependency), and actors (cp. set of typical goal modeling elements). Furthermore, the tool OME (Organization Modelling Environment) is a general, goal-oriented modeling and analysis tool that supports i* and modeling with the NFR Framework.

Several additions to the i* and NFR Framework have been suggested over the last years. [KDO07], for example, argues for adding hard goals to the NFR Framework, as non-functional goals are not always soft, but should also be phrased as measurable, rigid requirements. As can be seen, the NFR Framework provides a good way to model NFRs almost independently from functional requirements. The goal-modeling notation might be familiar to people who also structure their top-level requirements with goal models. Even though goal modeling has a strong academic community, until today, goal modeling has not found its way into the state of the practice. Furthermore, goal models are not the silver bullet for modeling all functional aspects of a software system. Therefore, even in the state of the art, the functional requirements specifications (especially on lower levels of abstraction) are modeled with features, structured natural language, scenarios, and UCs or UML diagrams. Only few approaches like [CL01a] relate the goal models to some UML models and UCs. It is obvious that one then has to use the NFR Framework goal

models in addition to the functional modeling techniques, or find another way to handle the NFRs. Using NFR goal models as an additional specification artifact tends to lead to low acceptance in industry for two reasons: 1) The goal models become large and, therefore, incomprehensible, and 2) the relationships of the NFRs to the functional requirements are difficult to observe. Therefore, NFR goal models are not a good candidate for the integrated specification of functional and non-functional requirements. The same holds for the treatment of NFRs in other related goal-modeling approaches such as Tropos [CKM01], which represents the graphs in a textual specification called "formal tropos specification". KAOS [Lam01a], [Lam01b] offers more functional models, such as an object and operation model. The object models and operational models are derived from the goal models. This means that if NFRs are captured in goal models, they are still separated from the objects and operational models. It is unclear how NFRs in the goal models would be systematically linked to elements in the object and operation model. As in other approaches, KAOS derives these models at a later stage in the requirements engineering process from the goal models instead of using the functional models as input for NFR elicitation.

Work on Integrating Goal Models into Functional Models

The work of Cysneiros and Leite [CL01a], [CL01b], [CL01c] investigates best the relationship between functional requirements and non-functional requirements. They use the NFR Framework to elicit and specify non-functional properties in the shape of goal trees. But additionally, they store information (primarily functional, but also non-functional information) in their LEL (Language Extended Lexicon), which is intended to be a vocabulary for the context of the system. In the LEL, words or phrases relevant for the specified field of the application are stored. They clearly state that they deploy two independent cycles for the functional requirements and for the non-functional requirements that need to be synchronized. Their main target is that when operationalizations (mainly new functionality) of NFRs are specified in the goal-trees, these should be integrated into the functional view. This is primarily done by extending the LEL to explicitly link LEL entries with functionality that results from goal trees. The additional information with regard to the operationalizations from the goal trees in the LEL can then be incorporated into UCs or sequence diagrams [CL01a] or UML class diagrams [CL01c]. In [CL01c], the authors clearly state that they do not use the UML specifications that describe the functional requirements for the elicitation of NFRs. They state that dealing with these aspects of NFRs can be quite difficult to be accomplished directly in UCs, scenarios, etc. For the NFR elicitation, they make use of the NFR Framework. They require that for each entry in the LEL, it has to be checked whether NFR goal trees exist or could apply and if so, they create a NFR goal graph (with the LEL symbol as related attribute) and incorporate the operationalizations into their LEL. The problem with this approach is twofold: The comparison regarding whether a goal tree applies is done on every LEL entry with only the top-level quality attribute (like Performance). On the one hand, this is

time-consuming as every LEL entry must be taken into account and, on the other hand, it is not precise enough, as they are only checked against the top-level quality attributes. Furthermore, the concept of the LEL does not represent a typical approach in state-of-the-art requirements engineering, thus being difficult to integrate into other modeling approaches.

[GS05] describe that they derive scenarios from goal models and later on they attach NFRs motivated from the goal models to a set of scenarios. No further information is given on how this is done, nor do they use the scenarios in a systematic way to elicit the NFRs.

The MOQARE Approach

The starting point of the MOQARE method (Misuse-oriented Quality Requirements Engineering) [HP08] is a functional description or draft of a planned or existing system, its business goals, and quality goals. From security requirements engineering, MOQARE adopts the general idea of identifying misuses [MF99], [SFO03], [SO00]. But MOQARE is designed to offer support for the elicitation and specification of all ISO 9126 internal and external attributes. Figure 19 shows a part of a MOQARE misuse tree, giving examples of the main concepts of MOQARE: *Business goals*

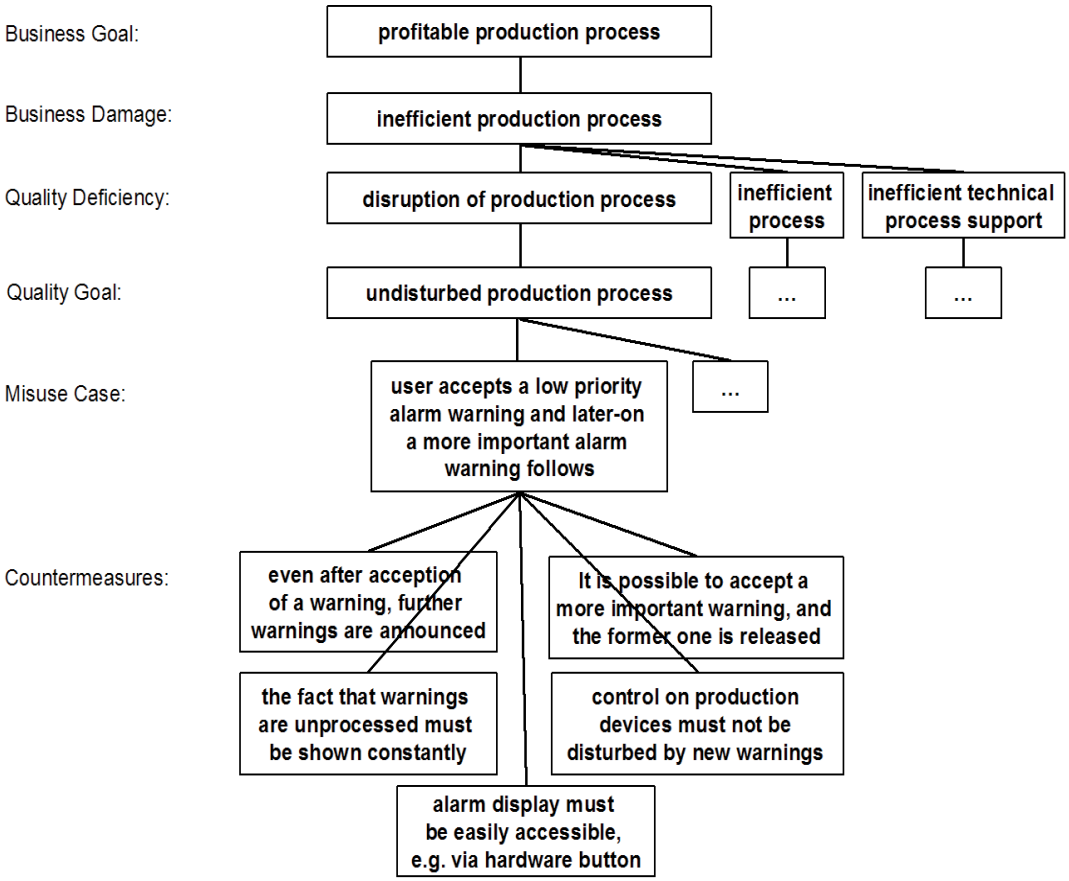


Figure 19: Example of a MOQARE misuse tree showing the main MOQARE concepts (figure taken from [HKD07])

might be threatened by *business damages,* which are caused by *quality deficiencies* of the system. A *quality goal* describes more specifically which part and property of the system supports the business goals. A quality goal is the combination of an *asset* and a *quality attribute*. Both have to be protected. The quality attributes used are those of ISO 9126. An asset is any part of the system. A *misuse case* describes a whole misuse scenario, including misuser, threat, and consequences (e.g., quality deficiency). A misuse is prevented, mitigated, or detected by *countermeasures*. The countermeasures can be new FRs, NFRs on FR, architectural requirements, or other quality goals. An interesting observation is that the term NFR does not occur in the MOQARE method. Rather than talking about detailed positive characteristics for a functional object (NFR), MOQARE uses the positive statement for the high-level quality goals and then determines countermeasures solely by eliciting and specifying negative characteristics (misuse cases, threats, misuser). As with the goal-oriented methodologies, no concrete definition for the target set of NFRs is provided, i.e., it is not clearly defined when the set of elicited NFRs is complete. This is also due to the fact that MOQARE does not use the complete set of functional requirements as systematic input as we do in our NFR methodology. Furthermore, MOQARE also proposes an additional, separate specification in addition to the functional specification. A detailed comparison between the NFR methodology elaborated in this thesis and the MOQARE method can be found in [HKD07].

The UMD Approach

[BDA04] provides "a structured framework for eliciting and organizing dependability needs". Similar to MOQARE [HP08], the approach uses a traced chain of concepts to describe situations to be avoided and phrase measures to prevent them. The basic concepts are Events, Issues, and Scope (see Figure 20). An event might cause an issue (that is, an undependable behavior, such as a failure) to happen. The issues are traced to the elements in scope they concern. The issues that can occur are classified with typical quality attributes like "Accuracy", "Response Time", etc. So a negated issue often corresponds to a non-functional requirement. The scope objects can be functional conceptual elements. UMD [BDA04] themselves state that the UMD "lets stakeholder specify the issues they don't want to occur. However, this doesn't suffice". So they also argue for a positive phrasing of non-functional requirements (de-

Figure 20: Concepts and examples of the UMD approach (figure taken from [BDA04])

pendability in their context). The concept "Measure" in the UMD approach relates to the metrics in this NFR methodology and the concept "Reaction" to the means in this NFR methodology.

NFR Methodology	NFR Framework	MOQARE	UMD
QA	Softgoal	Quality Goal	Classification for Issues
NFR	Softgoal	- (indirectly by negating the misuse)	- (indirectly by negating the issue)
Usage of FR Types	-	-	-
QA Types according to ISO and Functional Classification	-	QA Types according to ISO	QA Types according to ISO
Direct Relationship of NFR to FR	Relationship via Attribute in Softgoal	- (indirect relationship via relation misuse and asset)	- (indirect relationship via relation issue and scope)
Experience-based Quality Models	- (indirectly by set of goal model instances)	(indirectly via checklists)	Typical classifications for Issues
Project-specific Quality Models	Instances of goal models	Instances of misuse tree	-
Means	Operationalizations	Countermeasures	Reactions

Table 2: Comparison of concepts with state-of-the-art approaches

To facilitate the comparison between our NFR methodology and the three main NFR Frameworks presented in the state of the art, Table 2 shows the comparison of the different concepts of the approaches.

Other Approaches

[Ebe98] suggests dealing with NFRs in the same way as with FRs, but does not give advice on how to systematically do that. In consequence, this will not lead to a systematic elicitation of NFR.

[SM98] proposes using the QOC (question, option, criteria) notation described in [MYB+91] to define quality requirements. The requirements are captured in the criteria that are used to select the best (design) option concerning a question someone has with regard to to the future system. But this work again uses quality attributes to decide on design options rather than giving help in phrasing and documenting NFRs. [SM98] also proposes a process to be followed. It sets the quality attributes based on taxonomies, selects assessment metrics suggested by templates, and creates scenarios for the NFRs. The system model (e.g., architecture) is then assessed in terms of NFR scenarios. The promised benefits of this scenario-based approach are that a scenario explains what an NFR means to a user and a scenario acts as a definition of the

NFR because it provides an operational setting. The positive aspect of this setting is that in these scenarios, functional requirements are enriched with sometimes measurable quality information. Unfortunately, this happens unsystematically, i.e., there is no systematic guidance, nor do the NFRs seem to be highlighted in the scenario description.

[BH96] presents a tool-supported methodology for modeling quality attributes. This approach does not provide a detailed NFR elicitation. The focus of this method is on detecting conflicts among different quality attributes. The approach has a strong empirical basis (discussed in [IBR+01]) and is a relevant source for typical dependencies among quality attributes.

[BBF+01] and [Fra98] see the risk that quality attributes in standards are too high-level to model NFRs. Therefore, they introduce the language NoFun (acronym for NOn-FUNctional). NoFun shall serve as a language that provides a means to formulate the non-functionality in a precise way. NoFun allows binding NFRs to software modules, but only to system components (not functions, tasks, etc.). Basic concepts of the NoFun language described in [BBF+01] are:

- Three kinds of quality entity modules, i.e., characteristic, sub-characteristic and attribute modules. These modules correspond to the refinement hierarchy in quality model standards like ISO 9126.
- Behavior modules are assigned to each system component. Behavior modules are abstractions of software components and contain all relevant information for their quality evaluation. These behavior modules provide a mapping between the software components and the attributes modules from the quality entities.
- Quality Requirements are defined as restricting the values for the quality entities (see first bullet).

By evaluating the behavior modules, one can determine whether the quality requirements are met. The work of [BBF+01] and [Fra98] does not provide an algorithm or systematic help on how to elicit the quality requirements. According to [BBF+01], the UML proposes to annotate NFRs in UCs by using notes. [BBF+01] claims that this is insufficient and suggests a more ordered form. Thus, they propose using OCL and stereotypes to model Quality Entities, Quality Requirements, and Quality Behavior, but do not give sufficient detail to understand how this will be used.

3.1.3 Relationship Between FRs and NFRs

Several state-of-the-art approaches explicitly or implicitly link functional and non-functional requirements. As argued in Section 3.1.2, no approach provides detailed and comprehensive support, or joint specifica-

tion support for all NFRs. Typically, approaches such as [CL01a], [CNY+99] deal with functional requirements and non-functional requirements as two separate (independent) processes that are integrated later on. In this section, we present the state of the art with regard to approaches taking

- functional requirements or subsystem information as input to NFR elicitation
- functional requirements and NFR information as input to integrate them after a separate elicitation.
- functional requirements as output of NFR elicitation.

Functional Requirements as Input to NFR Elicitation

Few approaches use the functional requirements as input for NFR elicitation in a systematic way. [CL01c] and [CL01b] propose doing a pair-wise comparison of each quality attribute in their knowledge base with each symbol in the LEL (see also Section 3.1.2). But with this approach, too many comparisons need to take place, as the objects in the LEL are not classified as metamodel elements and therefore, the NFR elicitation cannot be focused beforehand. Furthermore, it is not specific enough, as the concepts in the LEL knowledge base are only the top-level QAs. The work of [CL01c] and [CL01b] anchors an NFR to a FR by using a type, which represents the link to the functional requirement it refers to.

Solms in [MS95] claim to have a method called Constraints Acquisition Technique (CAT) for dealing with Security, Safety, and Resilience Requirements starting from functional requirements and the environment. But what they call NFRs are rather quality attributes and the 3-D matrices used in their approach are more a prioritization of the quality attributes with regard to functional requirements and environment components. Questions that are asked there are, for example, "What is the importance of confidentiality for requirement 1 used by user 1?" So this approach seems to be more suitable for mapping quality attributes to functional requirements rather than for using the functional requirements as input.

[BMA02] use UCs to identify which QA have a cross-cutting nature in the sense that they constrain more than one UC. But [BMA02] do neither distinguish between QA and concrete NFR, nor do they use other functional conceptual elements or subsystems as input for NFR elicitation. Also [KOK04] use UCs as a source to identify NFRs, but they also do not use other functional conceptual elements or subsystems, nor do they systematically make use of quality models for the NFR elicitation.

[WDW08] use typical tasks of maintainers and quality models to identify maintainability requirements. Their approach seems limited to maintainability and only takes tasks of the type of the quality, i.e., maintenance tasks as input. Nevertheless, the idea is related to the derivation of NFRs of the type of User Task QAs in this NFR methodology.

[CNY+99] uses functional requirements for the breakdown of the NFR "Accuracy" to a more detailed level. But neither advice nor a systematic method is given across the QAs on how to use the functional information to derive detailed NFRs. Rather, [CNY+99] gives examples, e.g., example system elements in the NFR type catalogs.

Mapping NFRs to Functional Requirements

[LX99] also uses goals mainly to handle NFRs. They state that goals can have functional or non-functional content. They also state that a "non-functional goal is defined as constraints to qualify its related functional goal". [LX99] therefore indicate that non-functional goals should be directly related to functional goals. [KS95] present some work on integrating safety analysis into requirements engineering. In their work, they argue for a view-based documentation and attach non-functional requirements to certain viewpoints and, if applicable, to concrete functionalities (services). This is done in simple tables. Even though they did not introduce an explicit classification, the NFRs that were not attached to services were requirements on the project or development process.

In [Nix00], a work is presented that uses the NFR Framework for specifying performance requirements. Even if they do not deploy types of quality attributes or an explicit mapping of types of quality attributes to types of functional conceptual elements, they link performance requirements to business processes (i.e., to user tasks) and to the complete system. This mapping is done by putting the functional object in brackets behind the corresponding softgoal.

According to [JBR99], there are local NFRs (e.g., performance NFR) and global requirements such as security or reliability. As can be seen in the metamodel (see Figure 12), we agree with the distinction between local NFRs (constraining individual functional conceptual elements) and global NFRs (relating to the complete system), but this cannot be generalized for complete HLQA such as performance or security. This implies that NFRs will be documented at different documentation places.

[CL99] integrate QAs into an Entity Relationship (ER) model by adding them with a relationship to entities and relationships (see Figure 21). [BMA02] integrates QA into use cases and UML diagrames. [BMA02] does not distinguish between QA and NFR. Instead, they use a template to describe a QA like response time and include a field "where" and "requirements" where they specify which functional requirements are constrained by the QA.

The Squid Quality Process described in [BDK+99], makes use of quality requirements which are expressed by using metrics on various QA. They relate the quality requirements to objects called project portions. These project portions are project, not product artifacts, i.e., project activities, review points and deliverables. This makes sense as the focus of this approach is on expressing and achieving internal qualities, i.e., qualities that describe characteristics related to how the product was developed.

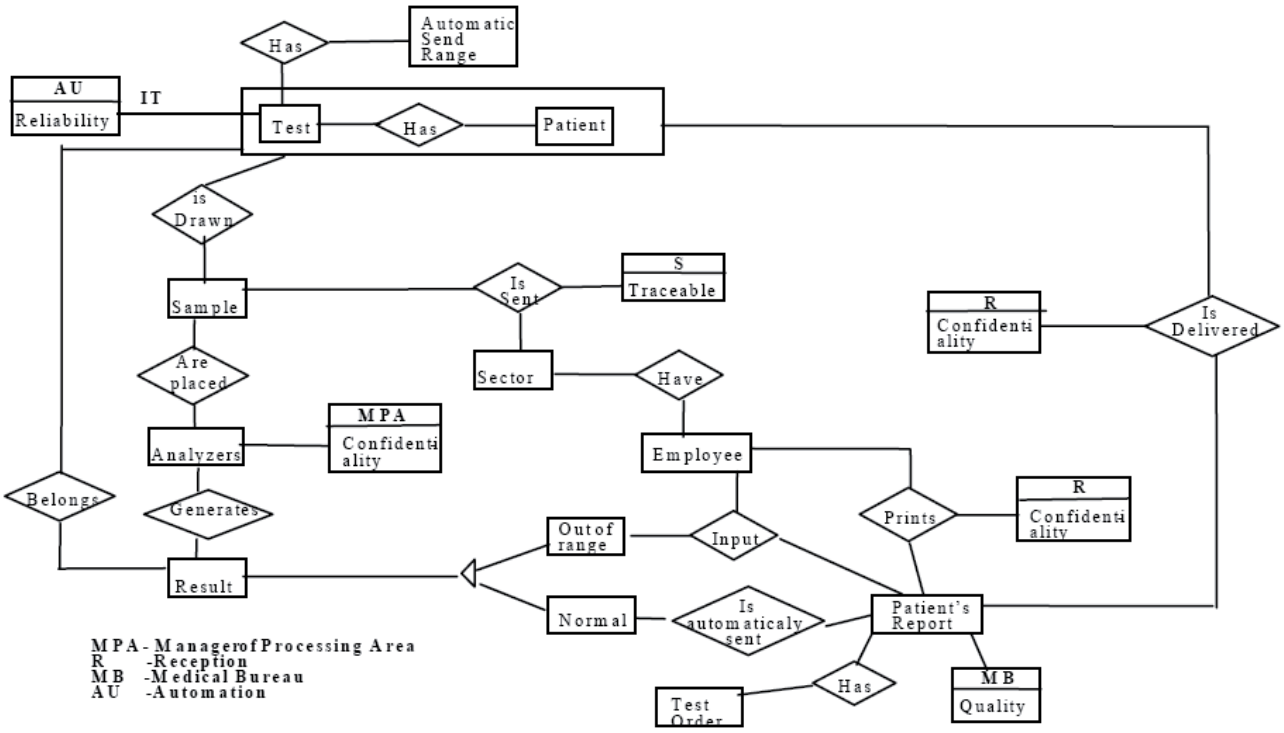


Figure 21: Example of annotating an ER model with quality attributes (figure taken from [CL99])

NFRs can and should also be linked to early quality assurance measures. [CMB08], for example, describe an approach to achieve complete traceability from goal-level to design level for the purpose of using this traceability information for impact analysis. They describe that quality goals should be related to design and code artifacts on the one side, and to quality assessment models such as ATAM or misuse cases on the other side.

Functional Requirements as Output of NFR

Several approaches like [Pas03] or [CNY+99] claim that NFRs are realized by introducing new functionality. [MRS+07] describe that NFRs can eventually evolve into a functionality of the system, into architectural decisions, or that they influence the development process. They also state that approaches dealing with NFRs as aspects solely work for the operationalized NFRs of the first type, i.e. the ones that result in functionality. Those kinds of functional requirements stay in a kind of "output" relation to the NFR. In [CL01c], [CNY+99] these new functionalities are part of the operationalizations and the authors include these resulting functional requirements in their goal trees. [CY98] focus on linking design decisions (design pattern) to goals. [CL01c] propose integrating the operationalizations of NFRs into the functional view. [LX99] use the "extends"-mechanism of UCs to create extension UCs that address non-functional goals. The extension UCs are also functional extensions that resulted from operationalized non-functional goals.

Most of the existing approaches that deal with NFRs also give some thought to modeling relationships between the different quality attrib-

utes (typically depicted by relationships in goal models). Also, some empirical studies on typical dependencies exist, such as the work of [IBR+01], which investigats the usefulness of the QARCC knowledge base.

Summary Integrated Specification

To summarize the current state of the practice and state of the art with regard to an integrated specification of functional and non-functional requirements, some major challenges to be addressed with this NFR methodology shall be pointed out:

- No clear distinction between QAs and NFRs: This makes it difficult to reuse the quality model information in a project-spanning way. Furthermore, some approaches attach means (countermeasures or operationalizations) directly to quite abstract quality attributes (see, for example, Figure 18) instead of enforcing the specification of a detailed, measurable NFR. Then the actual NFR is often not recorded explicitly. This situation makes quality assurance (e.g., the system test) difficult, as the implicit requirements (NFRs) cannot serve as a basis for test case derivation. Instead, the solution (means, operationalizations) is tested. Furthermore, having measurable NFRs enables the architect to systematically trade off the possible means/countermeasures. Having architectural alternatives related to quality attributes in a fuzzy way might cause the architect to make suboptimal decisions. Maybe the selected architectural means contribute to some qualities, but one cannot check which NFRs of the stakeholder are fulfilled completely or only partly. To enable substantiated decisions on architectural alternatives, measurable NFRs are needed.
- No end criterion for the elicitation and specification process: In the state of the practice and the state of the art, it is not possible to judge whether the NFR specification is complete, as no clear criterion is given for when the set of NFR is complete, i.e., the target set for NFR is not defined. Therefore, no formal end criterion can be defined for the elicitation and specification of NFRs, and it is not possible to formally determine when all NFRs are captured.
- Insufficient and non-systematic linkage between functional and non-functional requirements (see Figure 22): In the state of the practice, the non-functional chapters are mostly decoupled from the functional ones. In the state of the art, the strong focus on goal models or separate specification of NFRs introduces incompatibilities with the notations used in the functional world. Some of the state-of-the-art approaches incorporate mechanisms to link NFRs to data items, operations, processes, or components. The metamodel in Section 2.2.2 incorporates all these types of functional requirements and differentiates the quality attributes based on these functional conceptual elements and subsystems. None of the state-of-the-art approaches contains a similar classification.

In the NFR methodology in this thesis, the metamodel described in Section 2.2.2 introduced a clear distinction between QAs and NFRs and clear relationships between functional elements and quality elements. Completeness is defined for the intended set of NFRs. These are important concepts to enable the elicitation and specification of a complete set of NFRs.

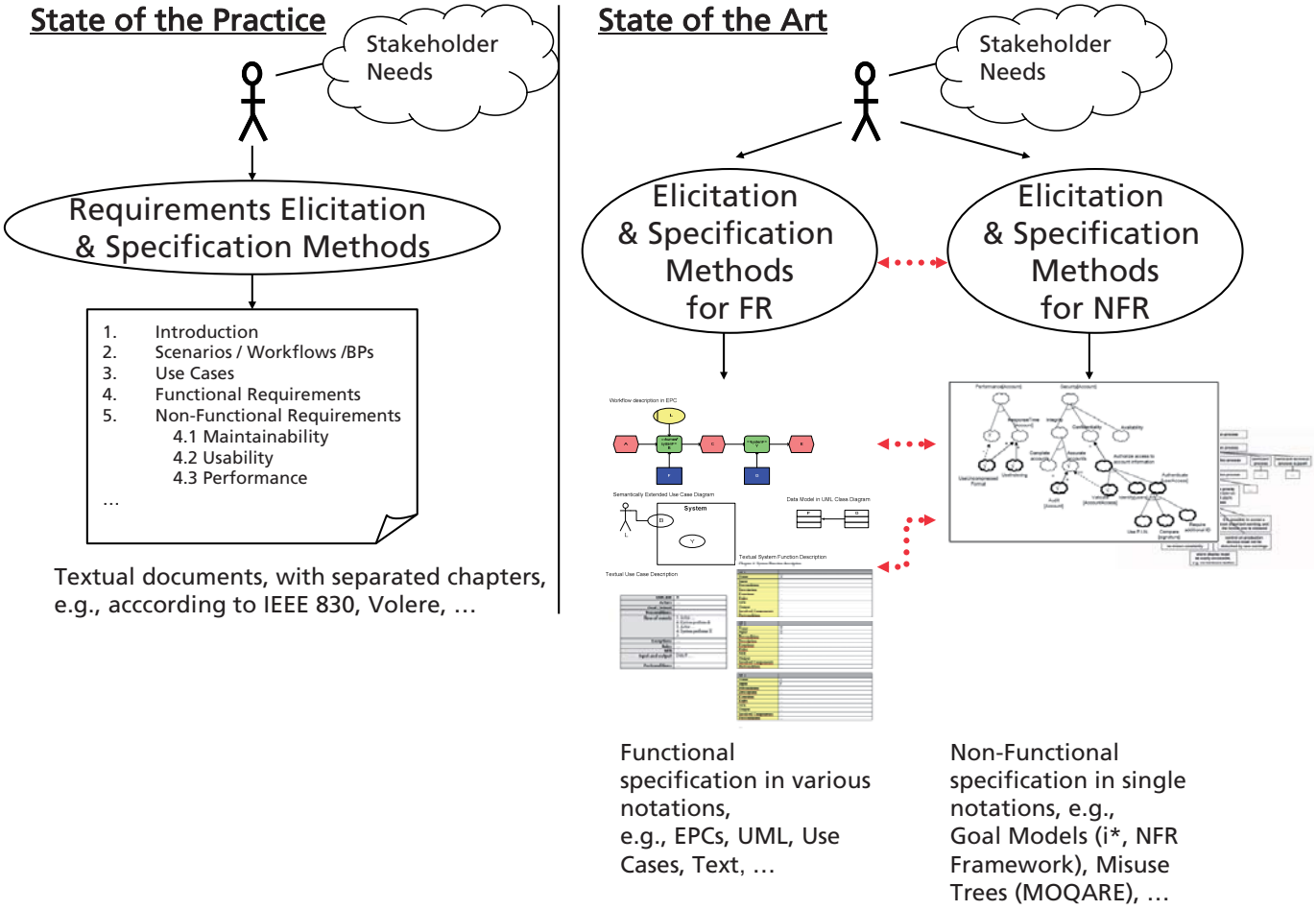


Figure 22: Non-integrated specification of FR and NFR in state of the practice and state of the art

3.2 Integrated FR and NFR Specification

As depicted in the metamodel (see Figure 12), functional and non-functional requirements and subsystems are closely related. Therefore, we argue for a joint specification of these product requirements rather than a separated one, as often proposed by the state-of-the-practice and state-of-the-art methods. [Ebe98] states that NFRs should not be separated from functional requirements in different sections of a specification: "While this is reasonable in few cases (development-oriented NFR), it can lead to serious fragmentation which reduces readability, particularly when functions and performance are split." The RUP [JBR99] also suggests specifying NFRs as close to UCs as possible. In contrast to the approach used in [CL01a], [CL01b], [CL01c], the concrete NFRs are attached to the functional conceptual elements and subsystems, but based on their classification and integrated with the functional specification, not in a goal tree (e.g., NFRs for user tasks in the user task specification, NFRs for system functions in the system function description). The joint specification in this NFR methodology shall be compatible with state-of-the-art specifications of functional requirements as well as with state-of-the-practice specifications of functional requirements like Volere or IEEE 1362. As the Task- and Object-oriented requirements engineering (TORE) approach [PK04b], [ADE+09] reflects a state-of-the-art requirements engineering approach for specifying functional requirements for interactive systems and, furthermore, TORE is compatible with state-of-the-practice requirements templates like Volere or IEEE 1362, it is a good candidate for serving as a basis to be extended with NFR information, from a state-of-the-art as well as from a state-of-the-practice point of view. Therefore, it will be introduced in the next section.

3.2.1 Relationship of Functional Elements to NFRs

Concerning the functional elements, the NFR methodology will base its concepts on the TORE approach [PK04b]. TORE provides a decision model for modeling interactive systems and is mainly based on a functional point of view, adding some usability aspects. Usablity aspects are the implementation of the paradigm of task orientation plus usability-specific decision points for decisions in the GUI design. The task orientation fits well with the task-driven nature of some NFRs and is therefore consistent with the task-driven elements of the metamodel. A mapping of general NFR types to a previous version of TORE can also be found in [Hae05]. In the following, an explanation of the mapping of the functional conceptual elements of the metamodel to the current TORE decision model [ADE+09] will be presented. Figure 23 gives an overview of this mapping.

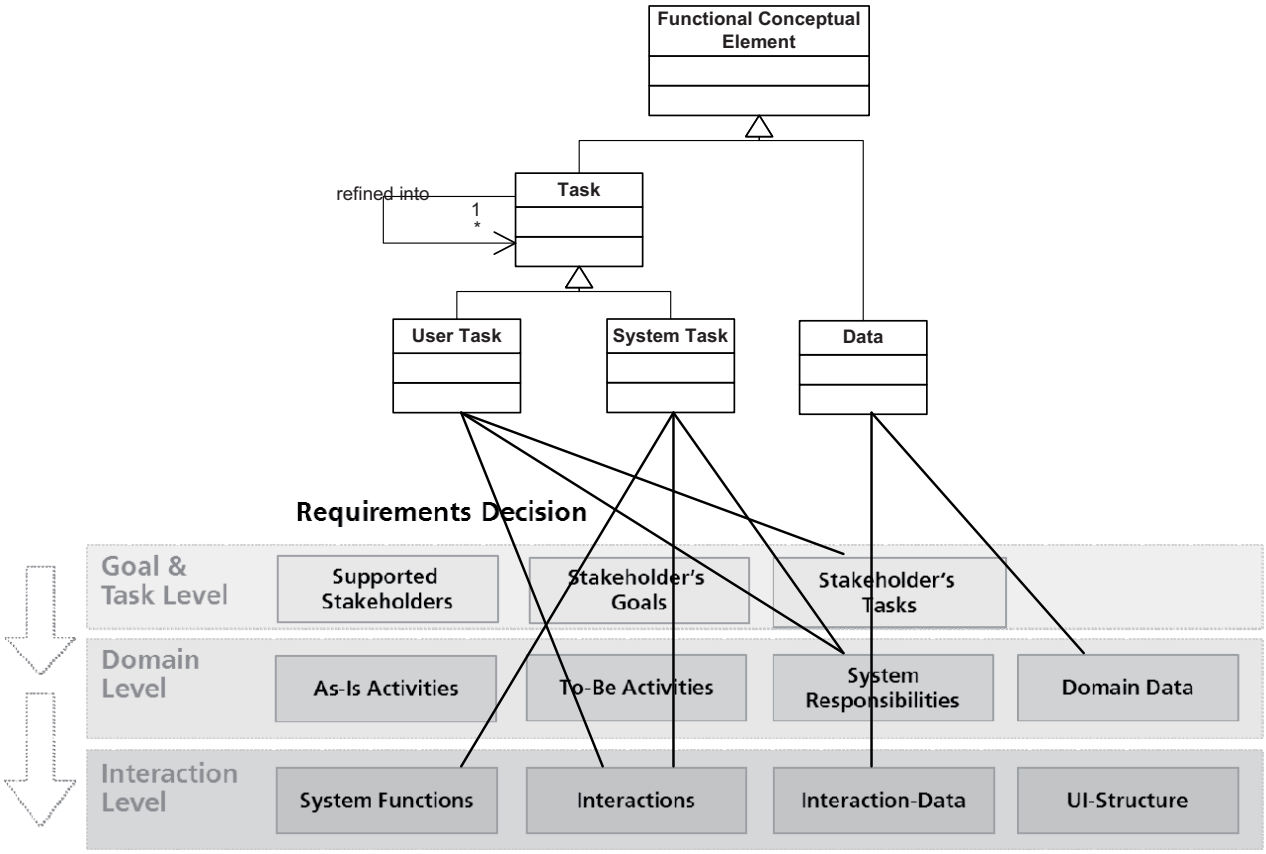


Figure 23: Mapping of functional metamodel elements to the TORE decision model

Goal and Task Level: As can be seen, the user tasks appear in TORE already at the task level. But this is just a naming of the user tasks without the task description. Additional elements on the task level are the stakeholders who will be supported by the SUD and their goals with the SUD.

Domain Level: The way the user tasks are performed today is described in the As-Is Activity descriptions. Therefore, these descriptions are not direct requirements for the system but background information. The way user tasks shall be performed in the future is described in the To-Be Activities. They describe the workflow of the user to perform his or her user task without determining which activities are performed by the user or the system. Typical notations used to record these decisions are business process modeling notations like EPCs [Sch99] or also UML Activity Diagrams [RJB99]. In the To-Be Activities, it is not clearly determined which activities are performed by the system or the user. Therefore, no links from system task and user task are drawn to the As-Is Activities. The System Responsibilities determine exactly which of the single activities within the To-Be Activity descriptions should be

- completely automated by the system (system functions): For these, system function descriptions are necessary. They correspond to the system tasks in the NFR metamodel.

- completely performed by a human (human functions): No product requirements originate from these activities.
- supported by the system (human-system functions): for these, interaction descriptions are necessary. They correspond to the user tasks in the NFR metamodel.

This decision is often recorded as semantically enhanced UC Diagrams or with stereotypes in the to-be activity descriptions.

Data requirements are already captured in the domain data and will be further refined on the interaction level. Typical notations used are glossaries, UML class and object diagrams, or ER diagrams. Domain and interaction data correspond to the data items in the NFR metamodel.

Interaction Level: For all system-supported activities, the human-system interaction has to be determined. An interaction description is typically recorded as a textual UC or as a sequence diagram with swim-lanes. The data used in the interaction is recorded as interaction data, usually as a refinement of the domain data. The decisions about the UI-Structure are for Usability purposes and out of scope for the mapping to the functional elements.

System Level: This level is not depicted in Figure 23, as all decisions on the System level are not related to the mapping to the functional elements of the metamodel. Further information can be obtained in [PK04b].

In order to describe the applicability of the NFR methodology for settings of various scale, we will start with a minimum set of sections for documenting requirements that is compatible with TORE, and then discuss how the template would change in case elements for workflow specifications should also be specified (for example if using for large business information systems). We describe template items that are directly related to elements of the metamodel to show the direct implications for the NFR methodology (cf. Table 3). Templates for requirements documents can comprise a lot more sections than the ones depicted below, see for example the section descriptions in [IEEE98a] or [RR99].

Conceptual Element from TORE	Conceptual Element from NFR Metamodel	Typical Section in Specification Template	Typical Notation used for Specification	Potential NFR Attachment
	Subsystem information	System Overview	Block diagrams, textual descriptions	NFRs for System QAs
User Tasks, System Responsibilities	User Tasks, System Tasks	Overview of operational scenarios	UC diagrams	NFRs of User Task QAs; NFRs of System Task QAs
Interactions	User Tasks, System Tasks	Operational scenarios; interaction descriptions	Textual UCs, sequence diagrams, flow diagrams	NFR of User Task QAs, NFRs of System Task QAs
System Functions	System Tasks	Feature lists, system function descriptions; functional requirements	Textual Function descriptions	NFRs of System Task QAs
Domain Data, Interaction Data	Data Objects	Data requirements; data model	Class diagrams, object diagrams, ER diagrams	NFRs of Data QAs
	Non-functional Requirements	Quality requirements; non-functional requirements	Textual description	NFRs for System QAs; over spanning NFRs
	Means	First solution ideas	Textual description	-

Table 3: Minimal set of conceptual elements required in templates

Thus, a minimal functional specification template that can serve as an appropriate basis for the NFR methodology needs to comprise the following chapters:

1. System Overview
2. Overview of Operational Scenarios
3. Operational Scenarios/Interaction Descriptions
4. Feature Lists/System Function Descriptions/Functional Requirements
5. Data Requirements/Data Model
6. Quality Requirements/Non-Functional Requirements
7. First Solution Ideas

Single User vs. Multi-User

This minimal specification template is usually used for single-user interactive systems like mobile phones, office products, graphics editors, etc. In contrast to that, for the specification of multi-user interactive systems

like large business information systems where many users use a system to cooperatively achieve an organizational goal (e.g., billing systems), workflow descriptions are often used as an additional specification element on a higher levels of abstraction to model the business processes. These workflow specifications can also be used for NFR elicitation.

In contrast to the user task descriptions, workflow descriptions are usually on a higher level of abstraction, often involving activities of more than one stakeholder or organizational role. The interface between workflow descriptions and user tasks according to TORE are then the tasks that one person will carry out. So in an ideal case, the workflow descriptions are detailed to the level where each activity in the workflow description is carried out by one role or is completely automated by the system. Then the user task descriptions would detail the user tasks for each role. As a logical consequence, eliciting NFRs in the workflow descriptions means that the NFRs for the User Task QAs can be elicited for all human-system functions carried out by one user (see beginning of this section for the classification). If a workflow activity is completely automated (System Functions), System Task QAs can also be elicited for this workflow activity. If workflows are documented with extended EPCs or other business process modeling notations, data objects are often allocated to the activities in the workflow. Then NFRs of Data QAs can also be elicited from the workflow descriptions. Table 4 visualizes this addition. The conceptual elements from the NFR methodology are introduced with the addition "indirectly", as they can only be mapped there if the workflow descriptions are enhanced with information from other decision points, i.e., the classifications resulting from the system responsibilities or the usage of data items from the domain data.

As a consequence, a typical specification template for addressing large-scale interactive systems would also have the workflow descriptions (but also many other sections not related to the metamodel, like business process hierarchies, organizational role descriptions, etc.) as part of the template. The variety of possibilities for documenting functional requirements for the functional conceptual elements can be seen as a enabler for NFR elicitation, but also imposes a major challenge for the inte-

Conceptual Element from TORE	**Conceptual Element from NFR Metamodel**	**Typical Section in Specification Template**	**Typical Notation used for Specification**	**Potential NFR Attachment**
As-Is Activities, To-Be Activities	Indirectly: User Tasks, System Tasks, Data Items	Business Processes	EPCs, Activity Diagrams	NFRs of User Task QAs, NFRs of System Task QAs, NFRs of Data QAs

Table 4: Extension of set of conceptual elements in templates

grated specification of FRs and NFRs. One example template that comprises all sections for a multi-user requirements specification with integrated NFRs can be found in Appendix B.

3.2.2 The Challenge of Integrated Specification

Standards like IEEE830 or IEEE1362 suggest having a separate or more than one separate chapter for NFRs. This strict separation of the non-functional requirements from the functional requirements is not appropriate for all types of NFRs. This way of separate documentation is appropriate for the NFRs affecting the system or subsystems, as they describe characteristics of the system itself or of subsystems, and it would be counterproductive to repeat these NFRs at each and every functional conceptual element. But NFRs that directly constrain a functional conceptual element like a user task, system task, or data item should be directly annotated at the corresponding user task, system task, or data item they are related to. Otherwise, readers of the document who are searching for information that is specific to one functional conceptual element will probably not realize the NFR that constrains the functional conceptual element. For example, the NFR affects the designer as the solution space for designing an implementation of the functional conceptual element is restricted. A tester might forget to write a test case for this specific quality characteristic of this functional conceptual element. There is one exception: If the same NFR holds for all instances of the functional conceptual element (like a default NFR for all UCs), we call these NFRs "over spanning NFRs" and we also put these NFRs into the separate chapter on non-functional requirements.

The annotation of NFRs to user tasks, system tasks, and data items imposes a major problem: How to deal with the situation that the same data items, user tasks, and especially system tasks can appear at different places in a requirements document? Figure 24 illustrates the problem based on a typical TORE specification example:

- The dashed lines illustrate where the same data object appears in different sections of a requirements specification. They appear in the EPC workflow description, the data model, the textual UC descriptions, and the textual system function descriptions.
- The solid lines illustrate where the same user task appears: in EPC workflow descriptions, in the UC diagram, and in the textual UC description.
- The dotted lines illustrate where the same system tasks appear: in the EPC workflow description, in the UC diagram, in the textual UC descriptions, and in the textual system function description.

The reason why these elements can appear more than once is often the fact that different views (e.g., structural vs. behavioral views in object

oriented specification of systems) and refinement steps are used to describe these system elements. For example in Figure 24, the name of a system task Y may first appear in the workflow description, reappears as a name in the UC diagram (within the system border), and is then described in detail in the textural system function specification.

One might think that in an ideal case, the NFRs are available at any occurrence of the functional element. But the disadvantage of this would be that NFRs would be overemphasized. Typically, each view in object oriented analysis should reveal specific information and not repeat information. Furthermore, replication of information usually makes the requirements management a difficult task, as in case of a change, information in many locations needs to be changed.

As the information included in the requirements specification is intended to be input for the subsequent development phases, the potential audiences and their preferences should determine where to finally document the NFRs. The potential users of the NFR information are:

- The requirements analyst, who checks the NFR information for conflicts (see Section 5.7). This person would need the information annotated at the joint functional object that is affected by the possibly conflicting NFRs.
- The architect, who uses the NFR information to make architectural trade-offs and decisions. In an ideal case, this person needs the information in different views.
- The developer, who uses the NFR information to develop a specific function, component, or service. This person would prefer the information to be annotated to the system function or component he or she has to implement.
- The tester, who will use the NFR information to derive test plans and cases for the product. This person would prefer the information to be annotated in the interaction descriptions.
- The reviewers, who want to see whether the integrated specification is acceptable with regard to quality characteristics like completeness. As with the architect, this person would need the information in different views.

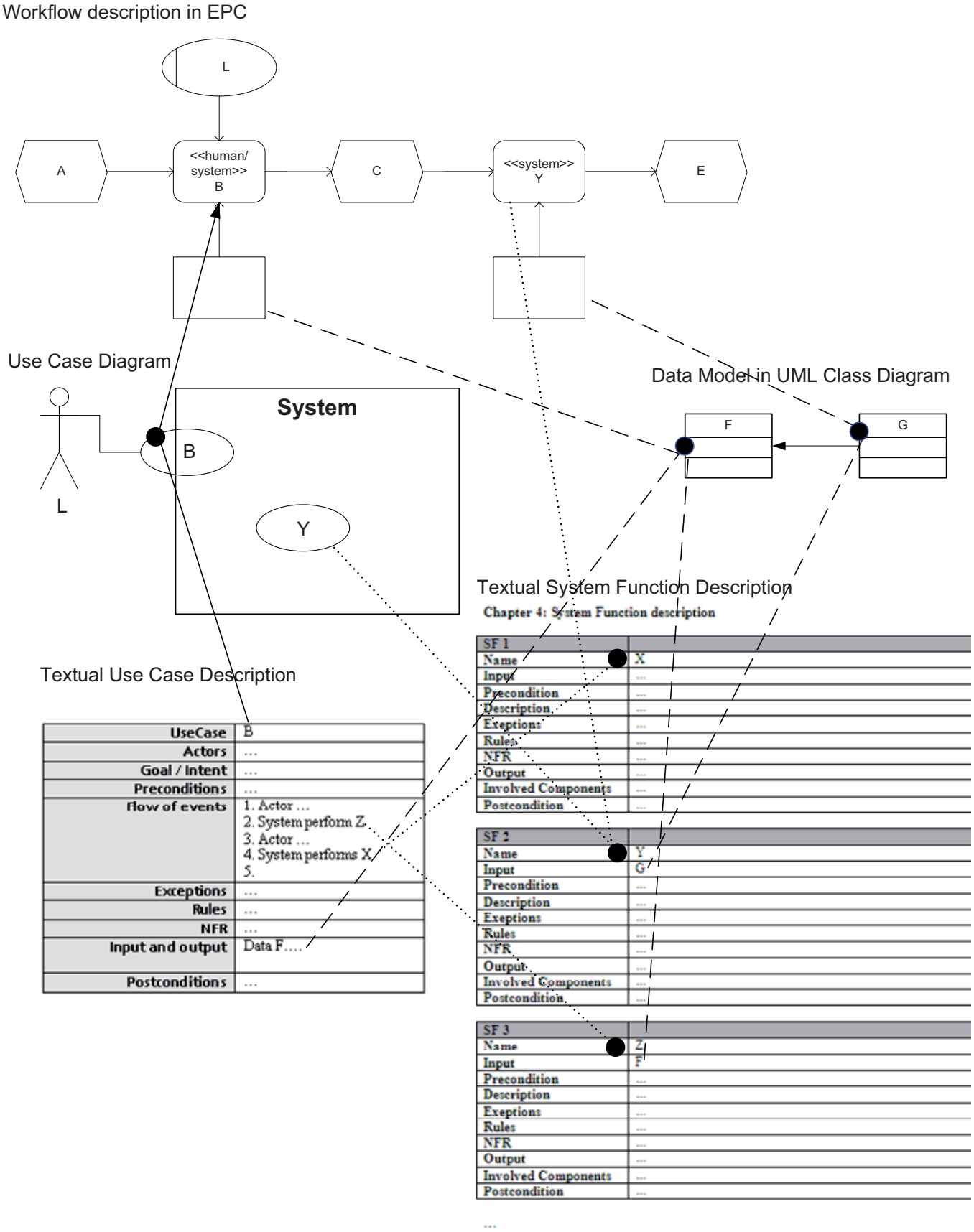


Figure 24: Illustration of the problem of Primary and Secondary Information Place

As can be seen from the list, the information need is different for the various roles and therefore, there is no ideal solution: The need differs from NFRs to be specified in every view to specification solely at the subsystems or interaction descriptions. If current tool support had a feature to enable and disable the NFR-information in the various views, this would definitely be a great help to address this challenge. But the development of such tool support is not the topic of this thesis.

The solution used in the NFR methodology is to introduce the concepts of primary and secondary documentation location:

- **Primary documentation location**: the location in the requirements specification where the main-information for a functional conceptual element is specified.
- **Secondary documentation location:** the location in the requirements specification where additional information for a functional conceptual element (like usage of the specified conceptual element) is specified.

In Figure 24, the filled circles at the end of the dotted, dashed, or solid lines represent the primary documentation location. The location where the main information for a functional conceptual element must be specified is therefore defined:

- For data objects, the data model is the primary information location.
- For user tasks, the UC diagram is the primary information location.
- For system tasks, the textual system function description is the primary information location.

The primary information location for all objects of interest should be well known by all readers of the document. This is a good practice that differentiates good from bad templates. If a reader is looking for a specific type of information, he or she must directly know in which section(s) to find this information.

The NFR methodology foresees that the NFRs for a certain functional conceptual element are at least specified at the primary information location.

If the NFRs are also specified at other locations, this can be beneficial and supportive for certain readers of the requirements specification. This should not be achieved by replicating the information. Alternative means are building corresponding views or linking the information to the secondary information locations.

UCs and UC descriptions are often used as a central element of specifications (see [JBR99]). [Gli07] also states that the RUP proposes documenting NFRs as close to UCs as possible and only put the general ones

into separate chapters. UCs connect the other pieces of information in the specification. In Figure 24, one can also see that the UCs are, besides the EPCs, the only specification element that integrates red, blue, and green lines. The EPCs also have this characteristic, but as they are on a higher level of abstraction, they connect fewer items than the UCs. Therefore, the UCs play an important role in connecting the other specification items. This is also the reason why they are intensively used for the elicitation of NFRs (see Chapter 4). If one decides to also visualize NFR information at a secondary information location, the textual UC descriptions are a good candidate.

Therefore, for the remainder of this thesis, the primary information location is set as depicted in Figure 24. When choosing a different requirements specification template or deleting sections in the template, the primary information location might change.

3.2.3 Resulting Locations for NFR Specification

With regard to the NFR attachment, the following types of NFRs are specified in different locations in the document:

- NFRs constraining User Task QAs are attached to the overview on user tasks, typically in the UC diagrams, and are, therefore, documented in a UC diagram section. The NFRs are either specified as notes in the UC diagram itself (see Figure 25) or as natural language statements directly below the UC diagram. If the NFR constraining the user task is identical for all user tasks, it is moved to the general chapter on non-functional Requirements.
- NFRs constraining Data QAs are annotated in the data models. If the data models are documented as UML class or object diagrams, one can also use the notes in the diagram or write the NFR below the diagram. If the data model is documented as a glossary, we propose attaching the NFR to the glossary item.
- NFRs constraining System QAs are specified in the chapter on the general non-functional requirements. The structure of this corresponds to the structure of the quality models, i.e., there is a hierarchical list of all System QAs. Below each System QA, there is a list of all subsystems.
- NFRs constraining System Task QAs are directly attached to the system function descriptions, e.g., in a separate field of the textual system function descriptions (see also textual system function description in Figure 24). If the NFR constraining the system task is identical for all system tasks, it is moved to the general chapter on non-functional requirements.
- Additional specification: As already depicted in the last section, it can be beneficial to additionally specify specific NFR information in the textual UC description as well. Of course, it is beneficial to specify the

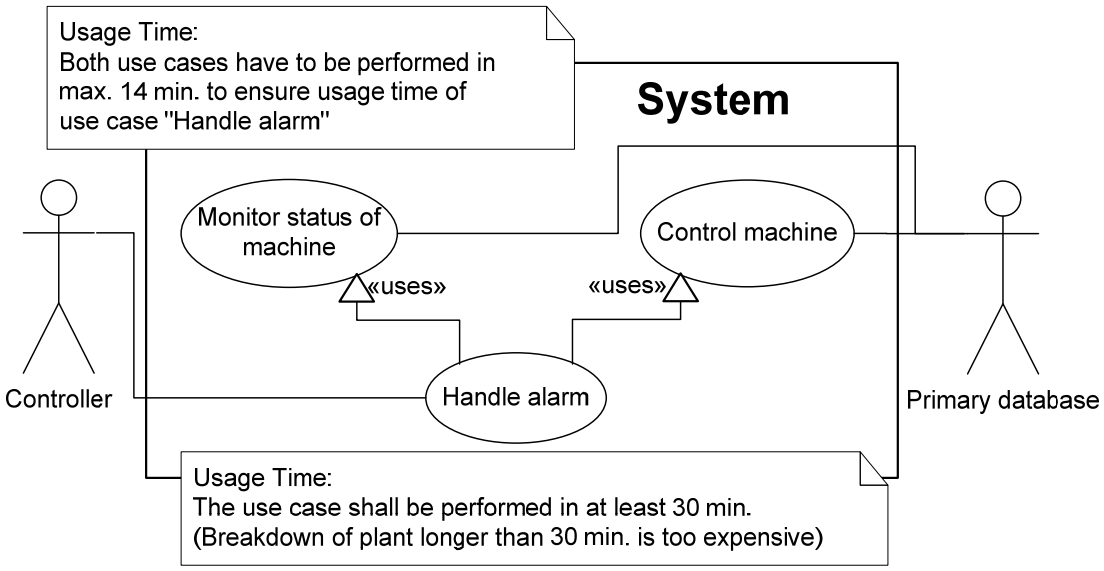


Figure 25: Example of annotation of User Task NFRs in a UC diagram

user task NFR that relate to the complete UC. But it is also beneficial to replicate the NFRs on system task QAs in the UC descriptions if the system function is used in the UC (see System Functions X and Z of UC B in Figure 24). This is especially true if the NFRs that are related to these system functions were created by a refinement of NFRs on User Task QAs (see Chapter 4 for details), as then the complete information for the UC is gathered in this one UC and can easily be analyzed for consistency. Therefore, the textual UC template is enhanced with a separate field for NFR specification.

A minimal template for specifying NFRs can be found in 3.2.1 and Appendix B. We want to emphasize that this is not a full-fledged requirements template but the minimal set that is needed for the NFR methodology, and that this set must be extended to meet the specific needs of each project. Work on specifying non-functional requirements with a focus on the workflow level can be found in [AD07a], [AD07b], and [ARD09].

For those cases, where the software model is completely based on UML models, also the UML QoS profile [OMG05] can be an option to annotate the various types of NFRs to the various types of UML models. An example on how NFRs and quality models can be specified with the help of the UML QoS Profile can be found in [RW07].

4 Elicitation of NFRs

In Chapter 2 and Chapter 3, the basis for the actual NFR elicitation has been created. The conceptual functional and non-functional elements were clarified by means of a requirements taxonomy, a metamodel, and definitions. Furthermore, the role of quality attributes, organized into quality models was explained. This chapter describes the main difference of this elicitation approach to existing approaches from the state of the practice and the state of the art. In order to avoid redundancy, a detailed survey of state-of-the-practice and state-of-the-art approaches will not be provided, as the relevant concepts of the approaches have already been described in Section 3.1. Furthermore, those approaches do not provide elicitation support similar to the one in this thesis. After explaining the difference to existing approaches, the necessary NFR elicitation algorithm will be explained. Furthermore, additional elicitation aids such as elicitation checklists will be introduced.

4.1 Main Difference to Existing Approaches

Before going into the details of the elicitation process, we want to emphasize the main difference of this elicitation approach to state-of-the-practice or state-of-the-art approaches that were described in Section 3.1.

Difference to State of the Practice

Existing state-of-the-practice approaches are typically based on brainstorming, i.e., the requirements analyst brainstorms with a customer about potential NFRs for their product. The template sections for NFR or quality models like the ISO 9126 are taken as a means for triggering certain NFRs. This process usually ends when no further NFRs come up. The problem of this kind of elicitation is that there cannot be confidence that all relevant NFRs were captured, as ***there is no objective end criterion for the elicitation process***. This is mainly due to the fact that the target set of NFRs is not precisely defined. Furthermore, the functional requirements are not taken as a systematic input for NFR elicitation.

Difference to State of the Art

Existing state-of-the-art approaches like [MCY+92] or [HP08] aim at a structured breakdown from high-level goals to more detailed goals that are comparable to NFRs, and eventually to countermeasures and operationalizations. The benefit of these approaches is that one has a much better overview to judge whether a breakdown makes sense and whether important segments are missing. But this elicitation process also has major drawbacks: First, as this elicitation process is decoupled from the

functional requirements[2], again the end criterion is when the experts think that the graph is complete, i.e., ***there is no objective end criterion for the elicitation process***. This is again due to the fact that the target set of NFRs is not precisely defined. Regarding completeness, [Yue87] states that goals "provide a precise criterion for sufficient completeness of a requirements specification; The specification is complete with respect to a set of goals if all the goals can be proven to be achieved from the specification and the properties known about the domain considered." But for a real-life project this proof cannot be provided. One reason for this is that complete traceability and ranking of the contribution power (completely fulfills a goal, contributes to a goal) of lower-level system requirements to the goal graphs is not feasible. Therefore, there cannot be confidence that the set of NFRs has been elicited completely. Second, the goal graphs are documented separately from the functional requirements, which is

- not viewed as a pragmatic solution in practice and therefore often not implemented. The NFR specification is then viewed as an additional specification rather than an inherently necessary part of the requirements specification.
- not a good basis for further development, as the NFRs in the goal graphs are dislocated from the related functional elements (see Section 3.2).

Therefore, the NFR methodology described in this thesis makes use of the relationship of NFRs to functional conceptual elements and subsystems. The functional conceptual elements and the subsystems are taken as input for the NFR elicitation. This information together with the information about the relevant quality attributes is used to algorithmically process the functional requirements in order to elicit a complete set of non-functional requirements. This algorithm has a defined end criterion as long as the input set of functional requirements and quality attributes is finite, which is a realistic assumption.

Repeatability and Controllability

The algorithmic nature also leads to higher repeatability of the elicitation and specification process. Everybody can judge which functional conceptual elements and subsystems were checked against which quality characteristics, leading to a degree of controllability that cannot be found in any state-of-the-art approach. This kind of controllability can also be used to focus the effort for the NFR elicitation (see Section 6.1).

[2] Current state-of-the-art methods like [MYK+92] integrate functional requirements into the goal graphs once it comes to the operationalizations, but often they do not include the functional requirements for the primary functionality of the SUD.

4.2 The Elicitation Algorithm

As the elicitation algorithm is crucial for this method, it is explained in the following. The basic procedure for this algorithm is as follows: We have the information about relevant qualities on the one side (QA_InScope) and functional conceptual elements (FR_E) and information on subsystems (SYS) that are characterized by the qualities on the other side. The basic idea is to have a complete pair-wise comparison between the elements in the quality dimension with the elements in the functional and subsystem dimension, asking whether NFRs exist for the quality - function/subsystem pair. The algorithm ends when the pair-wise comparison is complete. We know from the metamodel (see Section 2.2.2) that one quality attribute does not characterize all types of functional conceptual elements or subsystems; rather, each quality attribute has one specific object (user task, system task, data or system) it characterizes. Therefore, it is unnecessary to compare quality attributes of a certain type with elements of another type (like asking for Usage Time NFRs for a component). Figure 26 shows the comparison matrix that illustrates this situation. The gray areas are the ones that do not need to be covered by the algorithm as the quality attributes do not relate to the type of functional conceptual element.

Thus, the simplest version of the algorithm (compare Figure 27) would ask for NFRs in the linear way depicted in Figure 26.

Items to be compared	User Task QAs			System Task QAs			System QAs			Data QAs		
	QA_1	…	Qa_n	QA_{n+1}	…	QA_m	QA_{m+1}	…	QA_p	QA_{p+1}	…	QA_t
User Task 1												
…												
User Task 2												
System Task 1												
…												
System Task n												
System 1												
…												
System n												
Data Item 1												
…												
Data Item n												

Ask for NFRs

Figure 26: Items to be compared in elicitation algorithm, illustrated in a comparison matrix

For each $q \in EQAS$:

For each $f \in FR \cup SYS \mid (q, f) \in characterizes$:

Ask for NFR: IF (NFR=necessary) THEN please specify the NFR
Else: specify that no NFR exists
mark *qxf* as done

Figure 27: Simplified version of the elicitation algorithm

The simplified algorithm depicted in Figure 27 compares one by one the quality elements of a specific type with the related functional or subsystem elements. The algorithm has to be designed in a way that takes into account the specifics of the refine-relationship between user tasks and system tasks (see Section 2.2.2.1). As user tasks are often refined into system tasks, and so are quality attributes, the elicitation algorithm needs to address this fact. The assumption is that the probability for a customer to identify and state a corresponding NFR is higher if he or she thinks in refinements rather than being asked for a new NFR.

Figure 28 sketches the complete algorithm in a pseudo-code like fashion.

The three tasks in the algorithm are:

- asking for NFRs: The stakeholders decide and specify whether an NFR exists or not.
- refining an NFR: If a higher level NFR exists, the stakeholders decide and specify whether lower level NFRs exist or not.
- marking pairs as done: Pairs of functional conceptual elements/subsystems and elementary quality attributes get marked as done. This is done in order to allow an incremental approach: We know exactly which pairs have already been compared, so if the functional range is extended (e.g., due to changing requirements) or new qualities should be checked, the algorithm does not require the requirements analyst to repeat already performed comparisons.

For each $hq \in HLQA$:
    For each $utq \in QAUT$:
        For each $ut \in UT\_E$:
            Ask for NFR: IF (NFR=necessary) THEN specify NFR,
            ELSE specify that no NFR exists
            Mark *utxutq* as done
        For each $stq \in QAST \mid (stq, utq) \in refines$:
            For each $st \in ST\_E \mid (st, ut) \in refines$:
                Refine NFR: IF (NFR=necessary) THEN specify NFR,
                ELSE specify that no NFR exists
                Mark *stxstq* as done
            For each $st \in ST\_E \mid (st, ut) \notin refines$:
                Ask for NFR: IF (NFR=necessary) THEN specify NFR,
                ELSE specify that no NFR exists
                Mark *stxstq* as done
    For each $stq \in QAST \mid (stq, utq) \notin refines$:
        For each $st \in ST\_E$:
            Ask for NFR: IF (NFR=necessary) THEN specify NFR,
            ELSE specify that no NFR exists
            Mark *stxstq* as done
    For each $sq \in QASYS$:
        For each $ss \in SYS$:
            Ask for NFR: IF (NFR=necessary) THEN specify NFR,
            ELSE specify that no NFR exists
            Mark *ssxsq* as done
    For each $dq \in QADI$:
        For each $dq \in DI\_E$:
            Ask for NFR: IF (NFR=necessary) THEN specify NFR,
            ELSE specify that no NFR exists
            Mark *dixdq* as done

Figure 28: The elicitation algorithm

In order to illustrate the algorithm, based on Figure 26, we assume the following situation:

$$UT\_E = \{ut_i\},\ 1 \le i \le n;\ ST\_E = \{st_i\}, 1 \le i \le m;\ DI\_E = \{di_i\}, 1 \le i \le n;$$
$$SYS = \{sys_i\}, 1 \le i \le n$$
$$QAUT = \{QA_i\}, 1 \le i \le n;\ QAST = \{QA_i\},\ n+1 \le i \le m;$$
$$QADI = \{QA_i\}, p+1 \le i \le t;\ QASYS = \{QA_i\},\ s+1 \le i \le p;$$
$$\forall st_i, 1 \le i \le n : (st_i, ut_1) \in refines;$$
$$\forall QA_i, n+1 \le i \le n+2 : (QA_i, QA_n) \in refines$$

Figure 29 shows the comparison matrix including the processing steps of the algorithm:

1. In step 1, the algorithm asks for new User Task NFRs: It iterates for each User Task QA QA_1 to QA_n on all User Tasks ut_1 to ut_n.

2. In step 2, the algorithm asks to refine the User Task NFRs for User Task ut_1 for the child System Task QA_{n+1} for each refined System Task (System Tasks st_1 to st_n).

3. In step 3, the algorithm asks for new NFRs of the type System Task QA QA_{n+1} for the remaining system tasks st_{n+1} to st_m.

4. In step 4. the algorithm asks to refine the User Task NFRs for User Task ut_1 for the child System Task QA_{n+2} for each refined System Task (System Tasks st_1 to st_n).

5. In step 5, the algorithm asks for new NFRs of the type System Task QA QA_{n+2} for the remaining system tasks st_{n+1} to st_m.

6. In step 6, the algorithm asks for new System Task NFRs. It iterates for each remaining System Task QA QA_{n+3} to QA_m on all System Tasks st_1 to st_m.

7. In step 7, the algorithm asks for new System NFRs. It iterates for each System QA QA_{s+1} to QA_p on all Systems sys_1 to sys_n.

8. In step 8, the algorithm asks for new Data NFRs. It iterates for each Data QA QA_{p+1} to QA_t on all Data Items di_1 to di_n.

The algorithm is finished when each entry in this matrix is marked as done. To better understand the algorithm and its results, we refer to the example description of process activity P2.1 (see Section 5.6), which visualizes all comparisons done as well as the output of the algorithm.

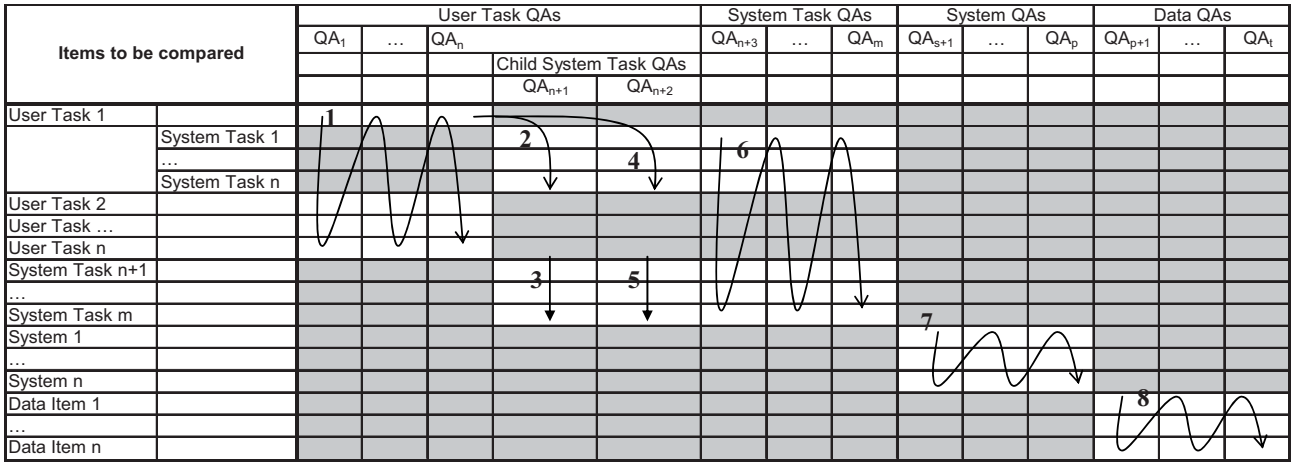


Figure 29: The different steps of the algorithm visualized in the comparison matrix

In the NFR methodology, the elicitation algorithm is "implemented" in two different ways:

- In the tool-supported version, it is implemented into the tool: The tool asks the user to either specify NFRs or to specify that no NFR exists (see Section 6.2).
- In the non-tool-supported version, it is incorporated into the checklists that guide the elicitation (see next section).

In both ways, the actual algorithm is executed in process activity P2.1 (see Section 5.6).

Elicitation Algorithm and Completeness of NFR_E

The major goal of this algorithm is to achieve a complete set of NFRs. Completeness was defined in Section 2.2.4 as: the set NFR_E is complete if there is a non-functional requirement expressed for each pair of elicited functional conceptual element or subsystem and corresponding elementary quality attribute, or there is no stakeholder request for it. The algorithm performs a pair-wise comparison between each functional conceptual element and subsystem and each corresponding elementary quality attribute. Therefore, the elicitation algorithm is designed in a way to guarantee a complete set of NFRs in the sense of the completeness definition under the assumptions stated in Section 2.2.4.

4.3 The Role of Checklists as an Elicitation Aid

The actual algorithm is either implemented in the tool support (see Chapter 6) or implemented in checklists that guide the manual elicitation process. In the manual setting, the requirements analyst uses the checklists in the NFR elicitation workshop with the customers. Figure 31 gives an example of such a checklist. The checklists are an additional means to

foster efficient NFR elicitation and are intended to ensure that the requirements analyst follows the algorithm and therefore achieves a complete set of NFRs. Other approaches such as [SHR09] also use questionnaires to prepare and guide the NFR elicitation, but the questionnaires do not incorporate a specific elicitation algorithm that guides the elicitation with the goal of reaching complete NFR specification.

Figure 30 illustrates the role of the checklists: For each quality model $QM \in QM\_InScope$, there exists one checklist that gives advice on how to proceed in an elicitation workshop. This advice also determines the corresponding functional conceptual element to iterate on. The concrete patterns for creating a checklist are explained in the next section. Following the advice results in NFRs issued by the customer, which are then documented according to Section 3.2 in the requirements specification. The dashed and dotted lines and boxes illustrate the connection between the elements in the three documents. For example, the QA Usage Time in the quality model leads to an advice in the checklist to specify Usage Time NFRs. Let's assume, the customer states two Usage Time NFRs for the two user tasks process bill and archive bill. As Usage Time is a User Task QA, the two NFRs are documented in the chapter overview on operational scenarios, which specify the user tasks in form of a Use Case Diagram.

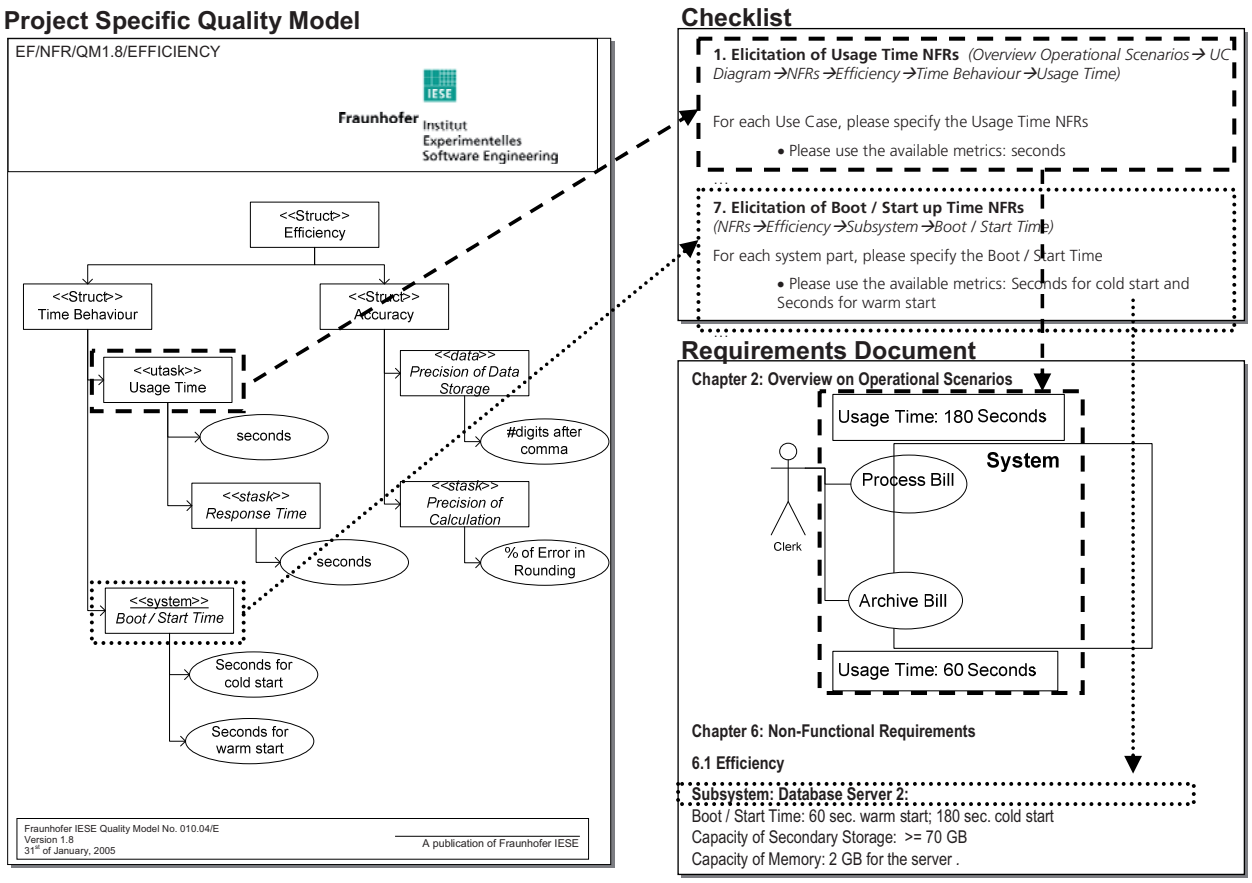


Figure 30: Checklists as mediator between quality models and NFRs

4.4 The Role of Reference Checklists and Templates

In order to make the preparation steps for the NFR elicitation more efficient, we suggest using reference checklists as well as a reference template that correspond to the QM_Ref rather than creating a checklist and template from the QM_InScope from scratch when a new project starts. This means the reference checklist and templates are then adapted to the changing quality models, i.e., changes from QM_Ref to QM_InScope are used to change the reference checklists and reference template. This means, for example, that if a $QA \in QA\_Ref$ is deleted because it is not needed for the current project, the corresponding checklist part and template part are also deleted from the checklist and from the template for the current project.

Usage of Sentence Pattern and Tool Support

The creation of the reference checklists is a straightforward, non-creative task. As the checklists get the quality model as input and the algorithm for elicitation is fixed, the checklist is generated by simple sentence patterns. In Chapter 6, we will see that the checklists can be automatically generated from the elements in QM_InScope. If the tool support for the elicitation step is used as well, the sentence patterns look a bit different (see right column in Table 5), as the tool has the capability of displaying each related functional conceptual element and subsystem, respectively. Of course, when using the Checklist Generation Tool, the reference checklists are not needed anymore. Furthermore, when also using the elicitation tool support, the checklist advices are built into the tool that guides the elicitation.

Please note that the sentence patterns presented in Table 5 are using the general terminology "user task", "system task", etc. from the meta-model. For a concrete checklist, we recommend instantiating these elements with the elements used for documentation (e.g., UCs, textual system function descriptions, etc.). An example of such instantiated sentence patterns can be seen in Table 8.

The sentence patterns for Data, System, System Task, and User Task QAs can furthermore be extended by adding a sentence inviting the user of the system to specify the NFR in a measurable way:

Classification	Sentence Pattern (Manual Process)	Sentence Pattern (Tool-supported Elicitation)
Structural QA	*No text, but a heading is introduced in the checklist to organize the advices*	*No text*
User Task QA	For each user task, please specify the *QA-Name*-NFRs	For this user task, please specify the *QA-Name*-NFRs.
"Child" System Task QA: The parent QA of this System Task QA is classified as User Task QA	For all user tasks: check if this user task has one or more NFRs of the type *Parent-QA-Name*: • If so, please refine these NFRs by specifying *QA-name*-NFRs at each system task refined from this user task. • Else: - For all remaining system tasks: please specify the *QA-Name*-NFRs.	• *For each user task, where NFRs of the type *Parent-QA-Name* were specified:* For the superior user task, the following NFRs were specified: *List of all NFRs on this User Task of the type *Parent-QA-Name**. For this system task, please refine these NFRs by specifying *QA-Name*-NFRs. • *For all other system tasks:* For this system task, please specify the *QA-Name*-NFRs.
"Stand alone" System Task QA: The parent QA of this System Task QA is classified as Structural QA	For each system task, please specify the *QA-Name*-NFRs.	For this system task, please specify the *QA-Name*-NFRs.
System QA	For each system part, please specify the *QA-Name*-NFRs.	For this system part, please specify the *QA-Name*-NFRs.
Data QA	For each data item, please specify the *QA-Name*-NFRs.	For this data item, please specify the *QA-Name*-NFRs.

Table 5: Sentence pattern for creating checklist advices

For each advice in the checklist, where the corresponding QA has one or more metrics attached (lets assume Metric_1, …, Metric_n), the following add-on can advise the requirements analyst to enforce the writing of testable and measurable NFRs:

- **Please use the available metrics**: Metric_1, …,Metric n-1 and Metric_n.

Figure 31 shows an example of such a checklist: the reference checklist for the $QM\_Ref_{Efficiency}$, using the instantiated sentence patterns for UC descriptions and system functions.

Elicitation of Efficiency NFRs

1. Elicitation of Time Behavior NFRs

1.1. Elicitation of Usage Time NFRs

For each use case, please specify the usage time

- Please use the available metrics: seconds

1.1.1. Elicitation of Response Time NFRs

For all Use Cases: check if this Use Case has one or more NFRs of the type Usage Time: If so, please refine these NFRs by specifying Response Time NFRs for each system function that is used in this Use Case.

For all other system functions: please specify the Response Time

- Please use the available metrics: seconds

2. Elicitation of Accuracy NFRs

2.1. Elicitation of Precision of Calculation NFRs

For each system function, please specify the Precision of Calculation

- Please use the available metrics: % of Error in Rounding

3. Elicitation of Time Behavior NFRs

3.1. Elicitation of Workload NFRs

For each system part, please specify the Workload

- Please use the available metrics: #jobs / time unit and # of supported user

3.2. Elicitation of Boot / Start Time NFRs

For each system part, please specify the Boot / Start Time

- Please use the available metrics: seconds for cold start and seconds for warm start

3.3. Elicitation of Shutdown Time NFRs

For each system part, please specify the Shutdown Time

- Please use the available metrics: seconds for shutdown

4. Elicitation of Resource Utilization NFRs

4.1. Elicitation of Workload Distribution NFRs

For each system part, please specify the Workload Distribution

- Please use the available metrics: % max

4.2. Elicitation of Capacity NFRs

4.2.1. Elicitation of Capacity of Memory NFRs

For each system part, please specify the Capacity of Memory

- Please use the available metrics: gigabyte, resource consumption of memory, cost of memory

4.2.2. Elicitation of Capacity of Network NFRs

4.2.2.1. Elicitation of Throughput NFRs

For each system part, please specify the Throughput

- Please use the available metrics: #data units / time unit, jitter and Mbit/s

4.2.3. Elicitation of Capacity of Processor NFRs

For each system part, please specify the Capacity of Processor

- Please use the available metrics: ghz, resource consumption of processor, and cost of processor

5. Elicitation of Accuracy NFRs

5.1. Elicitation of Precision of Data Storage NFRs

For each data item, please specify the Precision of Data Storage

- Please use the available metrics: #digits after comma

Figure 31: Reference checklist for $QM\_Ref_{Efficiency}$

Enhancing Understandability

During the NFR elicitation, the stakeholder who is asked for NFRs sometimes needs support in order to understand how to phrase NFRs. If the advice were merely phrased as: "For each system function, please specify the precision of Calculation", the stakeholder might not have an idea of how to phrase the actual NFR. The metrics that are attached to the checklist advices, such as "Please use the available metrics: % of Error in Rounding" provide such support. If this support is not enough to create a sufficient understanding for the stakeholder on how to phrase the NFRs, the checklist can additionally be enriched with example NFRs such as "A typical example for this kind of NFRs is: For this system function, a maximum of 0.5% error in rounding is acceptable".

Reference Template

Concerning the reference template, a standard template that minimally satisfies the elements depicted in Section 3.2 is taken. As the NFRs relating to data, user task, and system task are directly specified at the corresponding functional objects, the template must only foresee a possibility to do this (see Section 3.2). The only change in the reference template is with regard to the Structural QAs and System QAs that are in QM_Ref. For each of these QAs, the following change in the reference template should be incorporated:

- Structural QA: For each Structural QA, insert its own section into the general NFR chapter according to the hierarchy of QAs. E.g., if *Time behavior* is a structural QA below the Structural QA *Efficiency*, then the general NFR chapter of the requirements document template will have a section *Efficiency* with a subsection *Time behavior.*
- System QA: Insert a section with the name of the System QA below the Structural QA it belongs to. If, for example, the System QA *Precision of Data Storage* is below the structural QA *Accuracy*, the template will have a subsection *Precision of Data Storage* in the section *Accuracy*.

5 The NFR Elicitation and Specification Process

In Chapter 2, Chapter 3, and Chapter 4, the basis for the NFR elicitation and specification process was created. The term non-functional requirement was clarified by means of a requirements taxonomy, a metamodel, and a definition. The role of quality attributes organized into quality models was explained, and the basis for specifying the NFRs integrated with the functional requirements was created in Chapter 3. Chapter 4 described the elicitation algorithm and additional elicitation aids, such as elicitation checklists and reference checklists, which are derived from the quality models with the help of sentence patterns. Now that all necessary artifacts and elicitation aids have been introduced, this chapter describes how the artifacts from the last chapters are organized into a coherent process that enables effective and efficient elicitation of a complete set of NFRs. This process is typically run for each project where NFRs should be elicited for a SUD. Earlier versions of the process have been published, for example, in [DKK+05], [DOS04], [KDP+04].

5.1 Overview of the NFR Process

Figure 32 gives an overview of the NFR process. This process is divided into two process phases:

- Phase 1: the preparation phase: All activities needed for making an efficient NFR elicitation possible are carried out. First, the QAs relevant for this project are selected; in other words, the set HLQA is prioritized. Then, the reference quality models $QM \in QM\_Ref$ are tailored to the specific needs of the project, i.e., all $QM \in QM\_InScope$ (P1.2) are created. Once the quality models have been tailored, possible dependencies between the QAs in the quality models in scope are determined, i.e., one has to find all $q_1, q_2 \in QA\_InScope$ where $(q_1, q_2) \in influence$. This is done in P1.2. After the identification of possible dependencies, for the non-tool-supported version the checklists that contain the elicitation algorithm are derived in P1.4: One checklist is created for each $QM \in QM\_InScope$. Furthermore, in P1.4, the template for documenting the NFRs is adapted to the necessities of the new QA_InScope.
- Phase 2: the NFR elicitation and specification phase: In this phase, the actual NFR elicitation takes place (see P2.1). Either with tool support or by using the project-specific checklists, the requirements analyst elicits the NFRs together with the customer and records the resulting NFRs in the integrated specification (see Section 3.2). The ultimate

goal is to achieve *NFR_E=NFR* for the given qualities, subsystems, and functional conceptual elements in scope. This phase ends with a consistency check (P2.2) to ensure that the resulting set NFR_E is free of conflicts, i.e., $\forall\, n_i, n_j \in NFR\_E: (n_i, n_j) \notin conflict$.

- As depicted in the arrow at the bottom of Figure 32, there is a consolidation of the experience-based artifacts with the experience gathered in the current project.

In the following, each process activity P1.1 to P2.2 is explained in detail, describing the purpose of the process activity, the input to the activity, the output from the activity, the detailed activity description, i.e., how to perform the activity, and an example of its application.

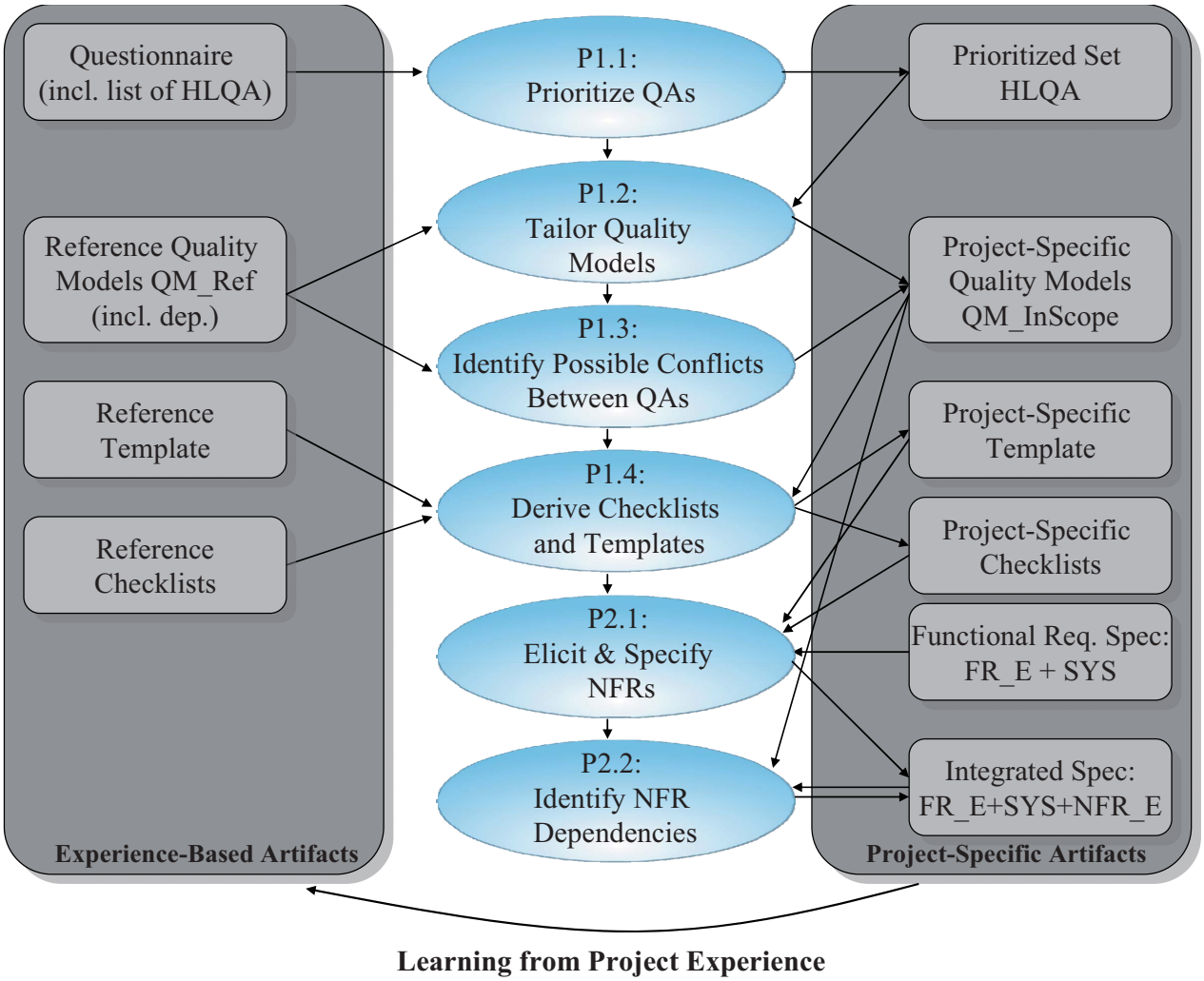


Figure 32: The NFR process

5.2 Process Activity P1.1: Prioritize QAs

Purpose: In large projects, it is not always desirable to elicit and specify the NFRs for all elements of HLQA, mainly due to effort restrictions. Therefore, the method foresees a possibility to focus elicitation and specification in the first activity of the methodology. This prioritization helps to focus the effort spent on NFR elicitation on the most important elements of HLQA. The goal is to reduce the number of NFRs that are the target of elicitation based on the QAs that are most important. In industrial projects (see also Chapter 7), it turned out that typically a maximum of 2 or 3 elements of HLQA were selected as the target of the NFR specification.

Input: The set HLQA expressing the set of QA that can be of interest to start the prioritization may vary, but it is usually similar to the following list of QAs that was already introduced in Section 2.2.3:

- Security
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Depending on the procedure selected for carrying out this activity, a predefined prioritization questionnaire can also serve as input for this activity.

Activity description: Basically, there are three possibilities for prioritizing HLQA:

1. Determination by using a prioritization questionnaire: Either a predefined prioritization questionnaire is used, or a new one is created for the list of $QA \in HLQA$. Creating a new one means that for each $QA \in HLQA$, derived statements that address sub-aspects of the corresponding QA are included in the questionnaire[3]. Representative customers receive this prioritization questionnaire and are asked to fill out the questionnaire. The filled-out questionnaires are then collected and evaluated to determine the order of the QAs. An example of a prioritization questionnaire and such an evaluation procedure can be found in [DKK+03].

[3] In order to obtain valid prioritization results, such a prioritization questionnaire should be standardized, i.e., the validity of the questionnaire must be statistically proven.

2. Determination by means of a stakeholder workshop using a common prioritization/voting technique: Representative customers are invited to a joint workshop. They are confronted with HLQA and are asked to prioritize the $QA \in HLQA$ by using a standard prioritization technique like the ones proposed in [Poh08].

3. Determination by deriving QAs from business goals: Another way to determine the priority of the QAs is to link the input list of $QA \in HLQA$ to business goals of the company that wants to develop the product the NFRs should be elicited for. In this case, the business goals need to be refined to the level of the HLQA input list. A method that uses such an approach is the MOQARE method [HP08].

The default case for the method developed in this thesis is the usage of a stakeholder workshop using a common prioritization technique.

Output: The output is a list of prioritized QAs from HLQA.

Example: For the X project, the requirements analysts used the standard HLQA list. They invited stakeholders (decision makers) from the customer organization to a joint prioritization session. They explained the meaning of each $QA \in HLQA$ by means of the ISO 9126 definition and some examples. Then each stakeholder was given 3 sticky points and was asked to stick the points on the highest-priority QAs from HLQA. The following result was achieved:

1. Efficiency
2. Security
3. Reliability
4. Usability
5. Maintainability
6. Portability

The requirements analysts for the X project therefore decided to first elicit the efficiency requirements for the SUD.

5.3 Process Activity P1.2: Tailor Quality Models

Purpose: For the most important QAs, the corresponding $QM \in QM\_Ref$ is tailored to the project specificities. This includes deleting obsolete child QAs but also adding new child QAs that are relevant to this project, but

were not included in the reference quality models. In this tailoring activity, domain experts from the customer and from the development company tailor each quality model to the needs of the project. As the completeness of the quality models is an important prerequisite for the later NFR elicitation (P2.1), this process activity is an important step in the NFR process.

Input: The list of prioritized QAs from P1.1 is taken as first input. How many highest prioritized QA are taken for this activity depends on the effort and time available for NFR elicitation (see also Chapter 6). The list of quality models that is selected for further processing is QM_InScope. Furthermore, the reference quality models (see Section 2.2.3) are taken as input for tailoring.

Activity description: Depending on the effort available for NFR elicitation, a number of high-priority QAs are taken from the list of prioritized QAs. Typically, this number corresponds to two or three QAs. For these QAs, the reference quality models are tailored to the specific project context. The most valid way to tailor the reference quality models is by performing a tailoring workshop with a moderator (expert in the NFR methodology) and various product stakeholders (including roles of the customer organization and of the development organizations). The following roles are good candidates to be invited to such a workshop:

1. Product Managers
2. Helpdesk Personnel
3. Support Personnel
4. Product Trainers
5. Architects
6. Testers
7. Product Customers
8. Product End Users

Typically, there will be one workshop for one QA. There are basically two ways to perform the tailoring in the workshop:

1. Alternative 1: Tailoring from scratch: For the tailoring, the moderator starts with the $QA \in HLQA$ (e.g., efficiency) as a trigger word and starts a free brainstorming on what this term means to the workshop participants. Using this procedure, the participants are to refine the term efficiency until they reach a measurable level. The moderator has two tasks: First, he or she collects and visualizes the quality model

resulting from the participants' discussion. Second, he or she continuously compares the current quality model with the reference quality model for the QA under discussion and suggests missing child QAs from the reference quality model to the group if the group does not raise new items during the discussion. The advantage of this alternative is that the group understands the created quality model as their quality model and the free discussion allows for new child QAs (that are not part of the reference quality models) to be discovered by the group. The disadvantage of this alternative is the long time that is needed for the free brainstorming.

2. Alternative 2: Tailoring by directly using the reference quality model: In this alternative, the moderator directly displays the reference quality model QM_Ref_{QA} and asks the group to comment on the model regarding obsolete and/or missing child QAs. The advantage of this alternative is that it is usually fast and the role of the moderator is less demanding compared to alternative 1. The disadvantage is the risk of forgetting important child QAs as the group trusts too much in the completeness of the reference quality model.

In recent industrial applications, we have made good experiences with alternative 1 and would recommend this unless it is not possible due to time restrictions or there is high confidence that the reference quality models contain all possible child QAs.

As an effort estimate for this activity, one high-level QA can typically be dealt with in a half-day to full-day workshop. The effort needed depends mainly on

- the number of people in the tailoring workshop,
- the alternative chosen for tailoring,
- the resulting size of the quality model,
- the ability of the moderator to focus the discussion on the quality model.

After creating these quality models, the method expert classifies the newly added QAs in the quality model according to the classification scheme depicted in Table 6 that corresponds to the metamodel (see Section 2.2.2):

Type of QA	QA Explanation
Structural QA	These QAs are just a means to structure the quality model. No metrics or means should be attached to this kind of QA.
User Task QA	These QAs are qualities that restrict tasks a user performs together with the system.
System Task QA	These QAs are qualities that restrict tasks a system performs without any further interaction with a human. The System Task QAs therefore restrict system functions.
System QA	These QAs are qualities that restrict complete systems or subsystems.
Data QA	These QAs are qualities that restrict data objects.

Table 6: Classification of QAs

Output: After performing this activity, the workshop(s) produces the project-specific quality models (QM_InScope), meaning that for each QA in scope for the project, exactly one quality model is created. In each quality model, every QA is classified according to the defined classification scheme (see Table 6).

Example: The requirements analyst in the X project invites the product managers and helpdesk personnel from the customer's organization as well as architects and testers from his or her own (development) organization to a tailoring workshop. The subject of this workshop is the tailoring of the highest prioritized element of HLQA, i.e., efficiency. The requirements analyst chooses alternative 2, the more efficient alternative, as the time for NFR elicitation is very restricted. Therefore, he or she starts the workshop by displaying Figure 14, which is his or her reference quality model for efficiency $QM\_Ref_{Efficiency}$.

The group makes the following modifications to the reference model:

- There is no need to talk about means at the moment: All means are removed from the model.
- The following QAs are irrelevant for the X project: Shutdown Time, Workload Distribution, Capacity of Network (Throughput and Network Topology), Precision of Calculation. Those QAs are removed from the model.
- The following QAs are missing: Capacity of Secondary Storage. This QA is added. The metric GigaByte is also added to phrase measurable NFR.

After the workshop, the requirements analyst classified the newly added QA "Capacity of Secondary Storage" as a System QA, as it directly affects the complete subsystem that is responsible for data storage. The resulting quality model is illustrated in Figure 33.

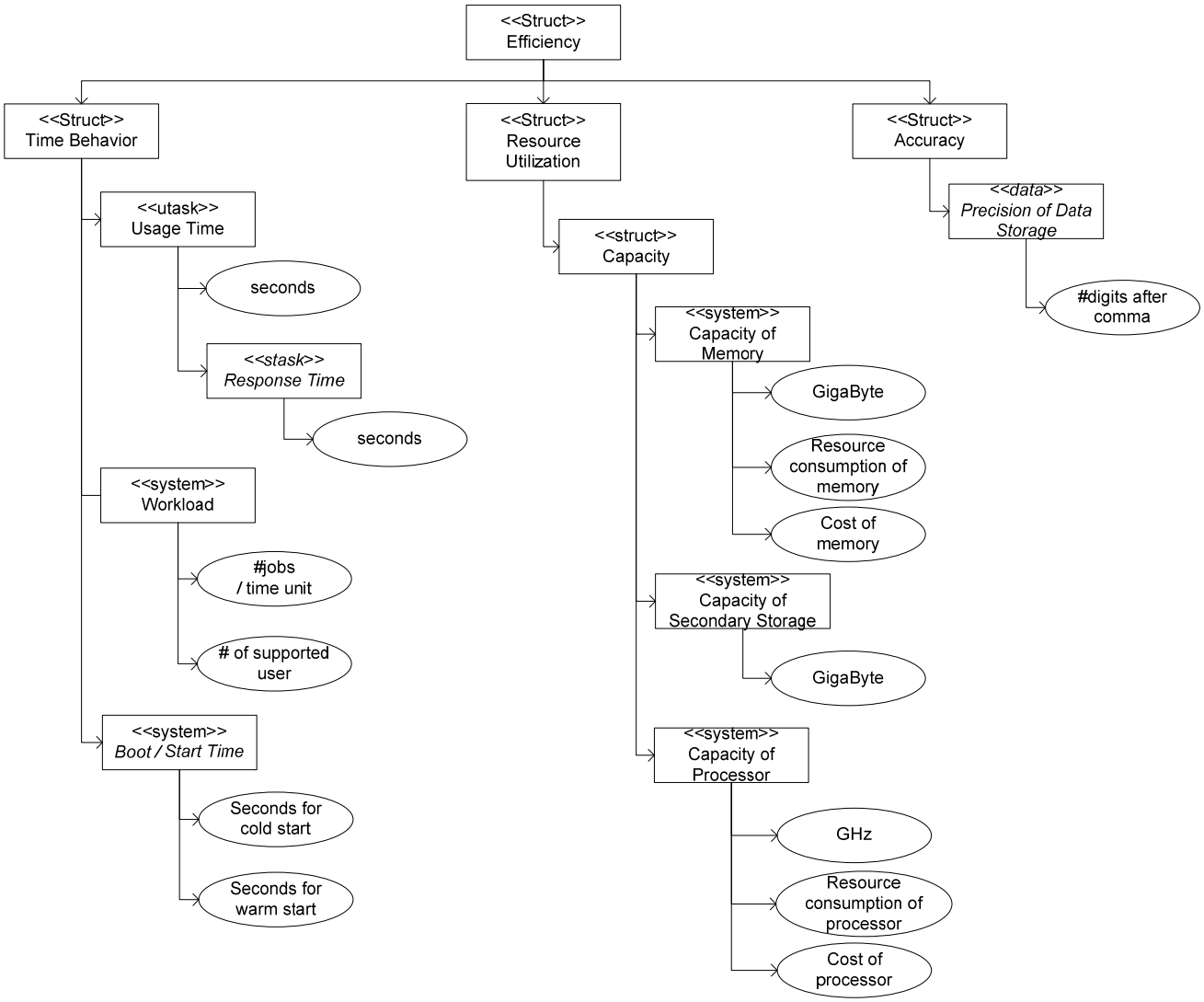


Figure 33: Tailored efficiency model: $QM\_InScope_{Efficiency}$

5.4 Process Activity P1.3: Identify Possible Dependencies

Purpose: This activity is only needed if P2.2 is also scheduled to be carried out in the project. After the tailoring of the quality models, this activity looks for possible dependencies that might exist between the QAs in the quality models within the scope of the project, i.e., it detects all $q_1, q_2 \in QA\_InScope \mid (q_1, q_2) \in influence$.

Input: This activity takes the project-specific quality models and the existing dependencies from the reference quality models as input.

Activity description: In this activity, the method expert performs two sub-activities:

1. Include existing dependencies in quality models: The method expert looks for influence dependencies in the reference quality models QM_Ref where the source and destination QAs for a dependency are also present in the project-specific quality models. For each such influence dependency, he or she includes this influence dependency in the project-specific quality models. In other terms: $\forall\, q_1, q_2 \in QA\_InScope, q_1', q_2' \in QA\_Ref: q_1 = q_1' \wedge q_2 = q_2' \wedge (q_1', q_2') \in influence \Rightarrow (q_1, q_2) \in influence$.

2. Check for new dependencies: The method experts evaluates whether QAs that have been newly introduced into the project-specific quality models in P1.1 might have dependencies on other QAs in QA_InScope. If so, the influence dependencies are included in the project-specific quality model. Depending on the know-how of the method expert, experts in the domain of the QA (e.g., security or performance experts) can or even should be involved in this activity. In other terms: $\forall\, q_1, q_2 \in QA\_InScope,\ q_1 \notin QA\_Ref,$ *Check if* $(q_1, q_2) \in influence$.

Output: The output of this activity are the project-specific quality models QM_InScope including all possible dependencies between the $QA \in QA\_InScope$.

Example: The requirements analyst takes the following input:

- The output of P1.2, i.e., the current status of $QM\_InScope_{Efficiency}$ (see Figure 33) and
- the dependency matrix from the efficiency reference quality model $QM\_Ref_{Efficiency}$ (see Figure 15).

As $QM\_Ref_{Efficiency}$ and $QM\_InScope_{Efficiency}$ are very similar, the requirements analyst decides to rather edit the $QM\_Ref_{Efficiency}$ than to start from scratch. Following sub-activity 1, he or she deletes those QAs that were deleted from the out-tree from the rows and columns of the dependency matrix as well (Shutdown Time, Workload Distribution, Throughput, Precision of Calculation). Following sub-activity 2, he or she adds a row and column for the newly added QA "Capacity of Secondary Storage".

	Usage Time	Response Time	Workload	Boot / Start Time	Capacity of Memory	Capacity of Processor	Capacity of Secondary Storage	Precision of Data Storage
Usage Time		↵	↵	↵				
Response Time			↵		↵	↵		
Workload					↓↑	↓↑		
Boot / Start Time					↵	↵		
Capacity of Memory								↵
Capacity of Processor								
Capacity of Secondary Storage								
Precision of Data Storage								

Figure 34: Intermediate result of P1.3

The resulting intermediate matrix can be seen in Figure 34.

The requirements analyst now has to check the new quality attribute "Capacity of Secondary Storage" and judge whether it might be in conflict with the other $QA \in QA\_InScope_{Efficiency}$. The cells where comparisons are needed are depicted with the black background color in Figure 34. Therefore, he or she has to do seven comparisons to fill out the seven undetermined fields. In case other $QA \in HLQA$ were also elements of QM_InScope (like maintainability, etc), the requirements engineering would also have to check for conflicts with all other QAs from the other models.

	Usage Time	Response Time	Workload	Boot / Start Time	Capacity of Memory	Capacity of Processor	Capacity of Secondary Storage	Precision of Data Storage
Usage Time		↵	↵	↵				
Response Time			↵		↵	↵		
Workload					↓↑	↓↑	↑	
Boot / Start Time					↵	↵		
Capacity of Memory								↵
Capacity of Processor								
Capacity of Secondary Storage								↵
Precision of Data Storage								

Figure 35: Final dependency matrix for $QM\_InScope_{Efficiency}$

As the requirements analyst is not an expert for efficiency, he or she asks the architect in his or her development team to assist in making this decision. The result of the comparison can be seen in Figure 35. Two new potential sources of conflicts were introduced: The higher the workload, the more secondary storage might be needed and the more precisely the data is stored, the more secondary storage might be needed.

5.5 Process Activity P1.4: Derive Checklist and Template

Purpose: Based on the project-specific quality models QM_InScope, the reference checklists and templates are tailored to the specific project context. The checklist are then used to guide the NFR elicitation and specification process P2.1. Therefore, in the non-tool-supported version of this methodology, the algorithm for the NFR elicitation is incorporated into the checklist.

Input: This activity takes the project-specific quality models, the reference template for specifying the NFRs, and those reference checklists that correspond to the prioritized QAs in scope for this project as input.

QA Type	Change in the requirements document template
Structural QA	For each Structural QA, insert its own section into the general NFR chapter according to the hierarchy of QAs. For example, if *Time behavior* is a structural QA below the Structural QA *Efficiency*, then the general NFR chapter of the requirements document template will have a section *Efficiency* with a subsection *Time Behavior*
User Task QA	No change to the template.
System Task QA	No change to the template.
System QA	Insert a section with the name of the System QA below the Structural QA it belongs to. If, for example, the System QA *Capacity of Memory* is below the Structural QA *Capacity*, the template will have a subsection *Capacity of Memory* in the section *Capacity*.
Data QA	No change to the template.

Table 7: Change of requirements document template based on QAs

Activity description: The method expert has two sub-activities:

1. Derive template:

   a. In this sub-activity, the method expert takes the reference template and deletes the sections for QAs that are not in QA_InScope.

   b. Furthermore, for each $q \in QA\_InScope,\ q \notin QA\_Ref$, he or she creates new sections for the newly identified child QA q in the template. Depending on the classification of the newly added QA q, the method expert performs the steps depicted in Table 7.

2. Derive checklists:

   a. Include existing advices in project-specific checklist:

   In this sub-activity, the method expert takes the reference checklist for each prioritized QA that is in scope for this project and deletes the advices for child QAs that are not in the project-specific quality models. After this sub-activity, all relevant advices from the reference checklists are included in the checklist version.

b. Create new advices:

In this sub-activity the method expert takes the checklist version from sub-activity 2.a and adds a new advice for each newly added child QA in the project-specific quality model. The creation of an advice is straightforward. It is dependent on the classification of the QA and is based on the sentence patterns depicted in Table 8.

Please note that the sentence patterns presented in this table make use of an instantiation of the generic sentence patterns in Table 5. The instantiation refers to the usage of textual UC description as documentation means for the element "User Tasks" and the usage of system functions as documentation means for "System Tasks" (see Section 2.2.2.1).

The checklist now contains all advices for the NFR elicitation step. Please note that in the tool-supported version of this methodology, the sub-steps 2.a and 2.b of this process step are automated by the Checklist Generation tool, see Section 6.2.1.

c. Revise order of checklist advices:

The checklist contains all necessary advices for the NFR elicitation, but the order is not optimized. Therefore, this NFR methodology recommends asking advices in the following order, according to the general elicitation algorithm depicted in Figure 28:

- Give advices for one high-level QA after another.

Classification	**Sentence Pattern (Manual Process)**
Structural QA	*No text, but a new heading is introduced in the checklist to organize the advices*
User Task QA	For each Use Case, please specify the *QA-Name*-NFRs
"Child" System Task QA: The parent QA of this System Task QA is classified as User Task QA	For all Use Cases: look if this Use Case has one or more NFRs of the type *Parent-QA-Name*: • If so, please refine these NFRs by specifying *QA-name*-NFRs at each system function that appears in the flow of events. • Else: - For all remaining system functions: please specify the *QA-Name*-NFRs.
"Stand alone" System Task QA: The parent QA of this System Task QA is classified as Structural QA	For each system function, please specify the *QA-Name*-NFRs.
System QA	For each system part, please specify the *QA-Name*-NFRs.
Data QA	For each data item, please specify the *QA-Name*-NFRs.

Table 8: Instantiated sentence pattern for checklist advices

- Then start with User Task QAs: give advices for User Task QAs,
- Continue with child System Task QAs
- Continue with remaining System Task QAs
- Continue with System QAs
- Continue with Data QAs

Output: First, one output of this activity is the complete project-specific template for the requirements specification for the functional and non-functional requirements. Second, for the non-tool-supported version of the methodology, this activity outputs the project-specific checklists that guide the NFR elicitation and specification process P2.1.

Example: The requirements analyst takes the reference checklist for efficiency (see Figure 31) and the tailored quality model for efficiency $QM\_InScope_{Efficiency}$ (see Figure 33) as input. Following subactivity 2.a, he or she takes all advices from the reference checklist except the advices for the deleted QAs Shutdown Time, Workload Distribution, Throughput, and Precision of Calculation. He or she adds a new advice for the newly added QA "Capacity of Secondary Storage", which is a System QA: "For each system part, please specify the Capacity of Secondary Storage-NFRs." As the metric Gigabyte is in the $QM\_InScope_{Efficiency}$, he or she adds the phrase "Please use the available metric: Gigabyte". As it is a System QA, it should be asked before the Data QA. Therefore, the requirements analyst moves the new advice up to position 2.1.3 on the checklist. The resulting checklist can be seen in Figure 36. In the template, the requirements analyst deletes the predefined sections on Shutdown Time, Workload Distribution, Throughput, and Precision of Calculation in the chapter on the general NFRs and adds a new section on Capacity of Secondary Storage in the chapter on the general NFRs, below the subsection on Capacity NFRs.

Elicitation of Efficiency NFRs

1. Elicitation of Time Behavior NFRs
   - 1.1 Elicitation of Usage Time NFRs
     - *For each use case, please specify the Usage Time*
       - Please use the available metrics: seconds
         - 1.1.1 Elicitation of Response Time NFRs
           - *For all Use Cases: check if this Use Case has one or more NFRs of the type Usage Time: If so, please refine these NFRs by specifying Response Time NFRs for each system function that is used in this Use Case.*
           - *For all other system functions: please specify the Response Time*
             - Please use the available metrics: seconds
   - 1.2 Elicitation of Workload NFRs
     - *For each system part, please specify the Workload*
       - Please use the available metrics: #jobs / time unit and # of supported users
   - 1.3 Elicitation of Boot / Start Time NFRs
     - *For each system part, please specify the Boot / Start Time*
       - Please use the available metrics: seconds for cold start, and seconds for warm start
2. Elicitation of Resource Utilization NFRs
   - 2.1 Elicitation of Capacity NFRs
     - 2.1.1 Elicitation of Capacity of Memory NFRs
       - *For each system part, please specify the Capacity of Memory*
         - Please use the available metrics: Gigabyte, resource consumption of memory, and cost of memory
     - 2.1.2 Elicitation of Capacity of Processor NFRs
       - *For each system part, please specify the Capacity of Processor*
         - Please use the available metrics: ghz, resource consumption of processor, and cost of processor
     - 2.1.3 Elicitation of Capacity of Secondary Storage NFRs
       - *For each system part, please specify the Capacity of Secondary storage*
         - Please use the available metrics: Gigabyte
3. Elicitation of Accuracy NFRs
   - 3.1 Elicitation of Precision of Data Storage NFRs
     - *For each data item, please specify the Precision of Data Storage*
       - Please use the available metrics: #digits after comma

Figure 36: Project-specific checklist for $QM\_InScope_{Efficiency}$

5.6 Process Activity P2.1: Elicit and Specify NFRs

Purpose: After determining the functional requirements that are in scope for the NFR elicitation from the complete set of functional requirements (FR_E) and subsystems (SYS), the set of NFR_E is elicited with the help of the project-specific checklist (or the elicitation tool) and documented in the integrated specification according to the project-specific template. Through pair-wise comparison of the functional requirements with the QAs, *NFR_E = NFR* shall be achieved for all QAs and functional requirements that are in scope for the NFR elicitation.

Input: For the non-tool-supported version, this step takes the project-specific checklist, the project-specific template, and the functional requirements specification (including FR_E and SYS) as input. For the tool-supported version, the tool takes the functional requirements specification according to the project-specific template as input.

Activity description: This process step contains four sub-activities:

1. Set the functional scope:

The first sub-activity of this process step is used to focus the effort for NFR elicitation. In the first step, the method expert sets the functional scope. This can be done in three ways:

a. Vertical scope selection: In this case, the scope is set by the level of abstraction that is the target of the NFR elicitation. The most common way to set the abstraction level is, for example, to set the UT_E as a target, but to not select the ST_E (which corresponds to the set of system functions) or the DI_E as target for the NFR elicitation. We recommend always selecting SYS as a target.

b. Horizontal scope selection: In this case, some functionalities and/or components are more important than others. The requirements analyst selects specific user tasks from UT_E and system components from SYS to be the target of the NFR elicitation.

c. Combination of vertical and horizontal scope selection: In this case, specific user tasks and components are in scope for the NFR elicitation, but only on a specific abstraction level.

2. Apply project-specific template to functional requirements specification:

Basically, we assume for this method that the functional requirements specification is already compatible with the functional elements of the project-specific template. Then, the method expert compares the pro-

ject-specific template and adds missing sections (these are usually the ones that concern the QAs in scope) to the functional requirements specification.

3. Elicit NFRs using the checklist/tool:

Concerning the non-tool-supported version of this activity, the requirements analyst invites representatives of the customer organization to an elicitation workshop. Adequate roles to be invited are the same as the customer roles already mentioned in P1.2. The requirements analyst who acts as moderator takes the checklist. His or her job is to stick to the checklist advices and navigate through the requirements document. As the checklist advices implement the algorithm for NFR elicitation in the non-tool-supported version, the elicitation process is straightforward. The moderator should carefully ensure the following points during the process:

- Stick to the process: Do not let the process drift based on new ideas emerging during the meeting. If NFRs for other QAs are issued by the workshop participants, note them as preliminary NFRs in the corresponding location of the integrated requirements specification (see sub-activity 4). Continue with the advices on the checklist.
- Use metrics: The workshop participants sometimes tend to neglect to use the metrics specified in the checklist and specify NFRs in a vague way. Enforce the usage of the metrics on the checklist or other measurable ones[4].
- Specify rationale: Especially during the first usages of this methodology, workshop participants tend to "overspecify" NFRs. This means they will raise NFRs that are not well motivated because they feel forced to give NFRs according to the checklist advices. This effect can be eliminated by always requesting a justification (rationale) for the NFR. Typical rationales in the context of NFRs are:
  - Several end users request the NFR.
  - This NFR is the result of several complaints on the hotline.
  - Our competitors offer similar qualities.
  - We want to surpass our competitors in this quality dimension.
- Watch out for over spanning NFRs: If the same NFR appears again, but restricts another functional conceptual element or system component, ask the customer whether this NFR holds for all elements of the same type of the functional conceptual element (e.g., for all user tasks or all system components). If so, mark the NFR as over spanning NFR.

[4] In case new metrics are suggested at the NFR workshop, these should be added to the project-specific quality models, the reference quality models, and to the checklist later on.

NFR of Type	Action for specifying the NFR
User Task QA	Specify the elicited NFR at the user task it belongs to. This can be done in the simplest way by annotating the UC diagram with a note that includes the NFR. Another possibility is to specify the User Task NFRs directly below the UC diagram structured according to the UC they belong to. Please note that over spanning NFRs are documented in the general NFR chapter of the document.
System Task QA	Specify the elicited NFR at the system task it belongs to. This can be done in the simplest way by directly writing the NFR into a specific field in the system function description (see system function template in Figure 24). If the system tasks are refinements of user tasks, i.e., if they also appear in the user task specification, we recommend to also specify (or link) the NFRs related to the system function in the corresponding user task. E.g., if UCs are used, one could specify the NFR in a separate NFR section of the UC (see UC in Figure 24). Please note that over spanning NFRs are documented in the general NFR chapter of the document.
System QA	Specify the elicited NFR in the corresponding[5] section of the integrated requirements document.
Data QA	Specify the elicited NFR at the data item it relates to. If the data items are described in a graphical manner (UML class/object diagrams or ER diagrams), the NFRs can be annotated similarly to the User Task QAs with notes in the diagram or placed directly below the diagram. Please note that over spanning NFRs are documented in the general NFR chapter of the document.

Table 9: Specifying NFRs in an integrated requirements specification

If this sub-activity is performed as described above, every functional conceptual element in scope is compared with every QA in scope that can be related to this functional conceptual element. The same holds for the system components. Under the assumption that the customers do not intentionally hide NFRs, the resulting set NFR_E will therefore be complete.

4. Specify NFRs in integrated requirements specification.

The NFRs elicited and validated in sub-activity 3 are then documented in the integrated requirements specification. This means that the NFRs are integrated into the extended functional requirements specification (see sub-activity 2). The actions described in Table 9 are performed during the workshop.

[5] "Corresponding" here refers to the fact that for the System QAs, separate sections were already created in activity P1.3.

For specifying the NFRs, the method expert should use a unique identifier for the NFR. The following scheme should be used to tag the NFRs:

If the QA is at the following location in the quality model hierarchy:

A→B→C→*QA-name*

the NFR will be tagged:

A-acronym.B-acronym*.*C-acronym*.*QA-name*.Number:*NFR-text*

As example: The first NFR for the QA Response Time (which is allocated in the quality model at Efficiency→TimeBehavior→Usage Time) would be tagged with

Eff.TB.UT.RT.1: …

The effort that is needed for the NFR elicitation depends on

- the number of functional items selected in sub-activity 1,
- the number of people involved in the elicitation workshop and their ability to agree on NFRs
- the experience of the participants and the moderator with the method (this mainly refers to sticking to the process and not opening up side discussions).

With the tool-supported version, the project-specific checklist in step 2 is not needed. The algorithm for NFR elicitation is built into the Elicitation Guide Tool, see Section 6.2.2.

Output: The output of this process step is the integrated requirements specification including functional (FR_E and SYS) and a complete set of non-functional requirements (NFR_E).

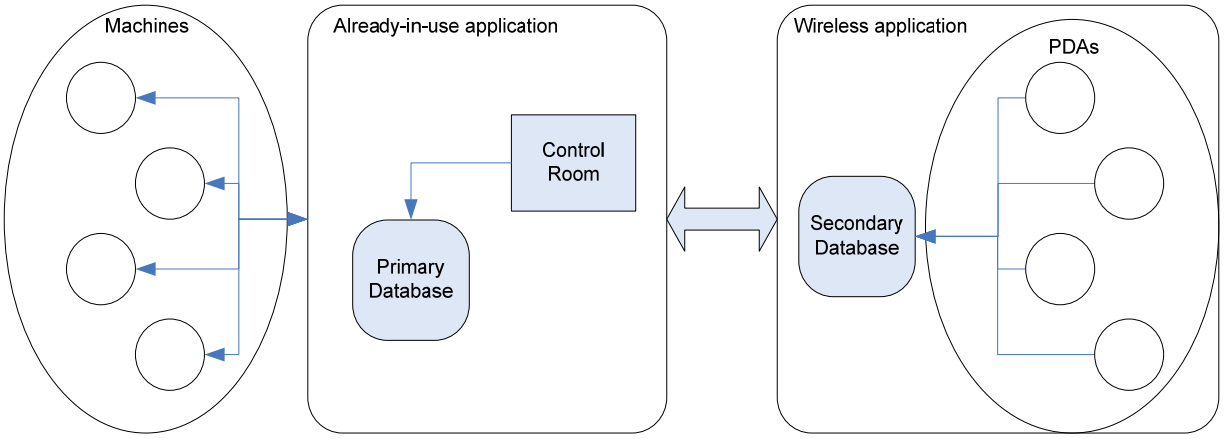


Figure 37: Physical components in the X project

Example: The requirements analyst for the X project invites the customer to a workshop. He or she takes the project-specific checklist (see Figure 36) and the functional requirements specification as input. As the time with the customer is limited to two hours, in a first step he or she sets the functional scope for the NFR elicitation. He sets the horizontal scope to all non-network elements of SYS, which are the primary database, the secondary database, and the PDAs (cf. Figure 37).

Furthermore, he or she selects the UC Handle Alarm and its included UCs as target of the elicitation, as this is the most crucial user task in the system (cf. Figure 38). In the vertical scope selection, the requirements analyst chooses UT_E, ST_E and SYS as the target of the elicitation.

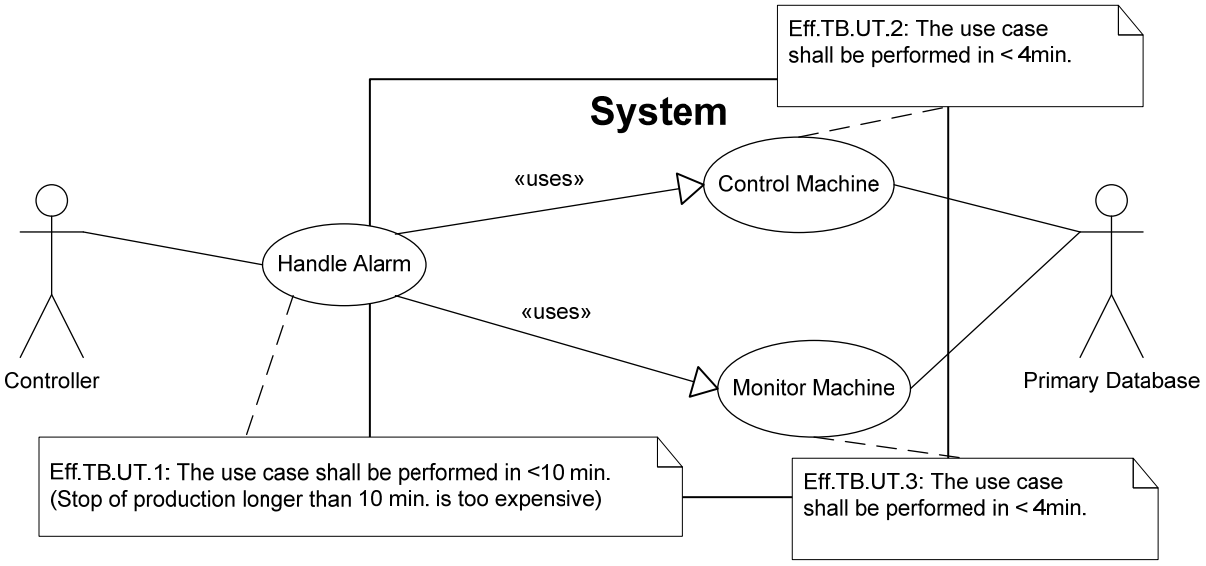


Figure 38: Target UCs for NFR elicitation in the X project

UseCase	Handle alarm
Actors	Controller
Goal	Actor removes a warning sent by a certain machine
Preconditions	…
Flow of Events	1. System regularly requests alarm messages from the first database 2. System shows alarm and where the alarm was produced. [Exception: Multiple alarms] 3. Actor acknowledges alarm to let other controllers know he/she is going to take care of it. [Exception: Another actor has acknowledged the alarm] 4. Actor moves to the machine following the path displayed by the system. 5. During the walk, the actor monitors the status of this machine with the help of the system (for details on the monitoring, see Use Case "Monitor Machine"). 6. Actor removes the problem by controlling the machine (for details on controlling, see Use Case "Control Machine"). 7. System sends control data to first database.
Exceptions	…
Rules	…
NFRs	Eff.TB.UT.RT.1: The response time for step 2 shall be < 5 sec. Eff.TB.UT.RT.2: The response time for step 7 shall be < 5 sec.
Postconditions	…

Figure 39: Use Case "Handle Alarm" in the X project

In the second step, the analyst extends the functional requirements specification with the missing elements for the NFR specification, i.e., he or she adds

- notes objects to the UC diagram (cf. Figure 38) for capturing the NFRs that constrain the user tasks in scope.
- a new template section "NFR" to the UC template (cf. Figure 39) to capture the NFRs that constrain the system tasks in scope.
- a new chapter "General NFR" to the functional requirements specification (cf. Figure 40) capturing the NFR for the subsystems in scope.

In the third step, the requirements analyst uses the checklist in the workshop to ask the customer for NFRs. He or she starts with eliciting the Usage Time NFR for the UCs. The customer states three NFRs for the UC Handle Alarm and its included UCs (see notes objects in Figure 38). Afterwards, the analyst asks the customer to refine the usage time NFRs into response time NFRs by using the textual UC definition. The customer refines the usage time NFR by stating two response time NFRs on the UC steps that are performed by the system (see Figure 39). The next checklist questions trigger the analyst to ask the customer for workload, boottime, and capacity requirements for all system components that are in

Chapter 4: General NFR
4.1: Efficiency
4.1.1: Time Behavior
4.1.1.1: Workload
Eff.TB.Wo.1: The processor of the first database must be capable of handling the data of 10 machines, which means aggregating the complete data of all ten machines every second (about 200 Kbyte of data per second).
Eff.TB.Wo.2: The PDA must be capable of having 5 programs open at the same time (rationale: assumed maximal number of programs that will be used by the user).
4.1.1.2: Boot / Start Time
Eff.TB.BT.1: The boot time for the primary database, the secondary database, and the PDA shall be <60 seconds.
4.1.2: Resource Utilization
4.1.2.1: Capacity
4.1.2.1.1: Capacity of Memory
Eff.RU.Ca.CaMem.1: The memory of the primary database should have 20% spare resources in a worst-case scenario (rationale: these are required for additions in the future) and it shall be <=512MB for the server (rationale: low cost for Hardware shall be achieved).

Figure 40: Added chapter for general NFR

scope. The customer states several NFRs for the system components which are also documented according to the fourth step in the integrated requirements specification (see Figure 40). The checklist part on accuracy NFR is skipped, as the data items were not included in the horizontal scope selection. The requirements analyst now has an integrated requirements specification for the X project.

5.7 Process Activity P2.2: Identify NFR Dependencies

Purpose: After NFR_E has been specified completely, this activity checks whether the set NFR_E is consistent or whether a set of NFRs in NFR_E is conflicting.

Input: The integrated requirements specification and the project-specific quality models (including the dependency matrices) serve as input for this step.

Activity description: This activity is divided into three sub-activities:

1. Check for intra-QA dependencies[6]:

In this sub-activity, one checks whether the set of NFRs specified for one and the same QA is consistent. This can be done by building a view on all NFRs for one QA and then do a pair-wise comparison for each pair of NFRs in this view. Experience shows that dependencies usually exist if the same functional object is affected. Therefore, it is reasonable to only compare NFRs of the same QA that are related to the same functional object or system component, respectively. For System Task NFRs, this can be achieved, for example, by navigating to the corresponding system function descriptions and comparing all NFRs of the same QA that are documented there.

2. Check for inter-QA dependencies:

Basically, all NFRs that are type of two QAs that are influencing each other (see Process Activity P1.3) should be checked for dependencies. This limits the scope of dependency checking from having to do a pair-wise comparison between all elicited NFRs to those where the QAs, the NFRs are type of, stay in conflict. So let

$$QA\_NFRexpressed=\{QA_i \in QA\_InScope \mid \exists\, n_j \in NFR\_E, (n_j, QA_i) \in isOfType\}$$

be the set of QAs where the elicited NFRs in the set NFR_E are type of. Then the algorithm for comparing NFRs would be:

For each $QA_a \in QA\_NFRexpressed$:

For each $QA_b \in QA\_NFRexpressed \mid (QA_a, QA_b) \in influence$

For each pair of NFR $NFR_i, NFR_j \in NFR\_E \mid (NFR_i, QA_a) \in isOfType \wedge (NFR_j, QA_b) \in isOfType$:

Compare and check whether a conflict dependency exists.

If a dependency is detected, this pair is marked as conflicting and must be resolved in step 3.

[6] Please note the use of terminology: We distinguish between intra-QA dependencies (i.e., dependencies between NFRs that are of the type of the same QA) and inter-QA dependencies (i.e., dependencies between NFRs that are of the type of different QAs). The dependencies between QAs of different quality models are called inter-quality model dependencies; the dependencies between QAs in the same quality model are called intra-quality model dependencies.

One could now ask the question whether a pair-wise comparison is sufficient for identifying possible dependencies. Maybe a set of NFRs of a size > 2 stays in conflict, but no pairs of NFRs inside the set. Then the procedure proposed here would not be able to detect this kind of conflict. There are two reasons why this method does not provide a mechanism for identifying such conflicts:

- It is not yet proven that such a situation may occur in practice. In all applications of the method and industrial settings we found a situation where when a set of NFRs with more than 2 elements was in conflict, this was always detectable by finding conflicts in the pair-wise comparison.
- Extending the already time-consuming pair-wise comparison to a check where all combinations have to be checked (see also Section 6.1.3) would increase the effort needed to a completely impractical amount.

Experience has shown that checking for inter-QA dependencies is a non-trivial task. This is due to the large number of comparisons that are needed to completely check all possible NFRs that might be in conflict, even though the number of comparisons is reduced by focusing on those NFRs where the corresponding QAs stay in conflict. Therefore, for this activity, tool support would be needed, but this is beyond the scope of this thesis. This tool support could bring the candidate pairs of possibly conflicting NFRs to the attention of the requirements analyst and ask whether the two NFRs really stay in conflict. This would take the burden of searching for the candidate pairs off the requirements analyst.

3. Resolve dependencies: Basically, there are two ways for resolving conflicts:

   a. One NFR dominates the other: The less important NFR is deleted or made less restrictive so that both NFRs can coexist and a solution can be constructed that will fulfill both NFRs.

   b. A compromise is made: Both NFRs are kept and both are made less restrictive so that they can coexist and a solution can be constructed that will fulfill both NFRs.

Variant a is often chosen if the elements of HLQA of the two QAs, of which the NFRs are a type of, have a significantly different priority (e.g., one NFR constrains the QA Analyzability, which belongs to the low-priority element of HLQA "Maintainability", and the other NFR constrains the QA Usage Time of the high-priority element of HLQA "Usability").

Output: After this activity, the integrated requirements specification contains a conflict-free set NFR_E, i.e., $\forall\, n_i, n_j \in NFR\_E: (n_i, n_j) \notin conflict$.

Example: The requirements analyst takes the integrated requirements specification and performs step 1 of this activity: The intra-QA dependency check. For this activity, he or she checks whether the three Usage Time NFRs are in conflict, which is not the case. Then he checks the set of two Response Time NFRs and afterwards the set of three workload NFRs, which are also conflict-free. In the second step of the P2.2 activity, the analyst checks for inter-QA dependencies: As only efficiency NFRs were in scope for the elicitation, he or she can focus on intra-QM dependencies. He or she refers to the project-specific dependency matrix (see Figure 35) and has to perform the following checks:

- Usage Time NFRs vs. Workload NFRs
- Usage Time NFRs vs. Boot Time NFRs
- Response Time NFRs vs. Workload NFRs
- Response Time NFRs vs. Capacity of Memory NFRs
- Workload NFR vs. Capacity of Memory NFRs
- Boot Time NFRs vs. Capacity of Memory NFRs

The analyst does not have to check Usage Time NFRs vs. Response Time NFRs, as this is already assured in the constructive refinement of Response Time NFRs from Usage Time NFRs. Since the analyst wants to make sure that the memory is sufficient for handling the amount of data, he or she only investigates the following pair of requirements in more detail:

- "Eff.TB.Wo.1: The processor of the first database must be capable of handling the data of 10 machines which means aggregating the complete data of all 10 machines every second (about 200 Kbyte of data per second)." and
- "Eff.RU.Ca.CaMem.2: The memory of the primary database shall be <=512MB for the server".

He or she asks one of the architects to make sure that the primary memory suffices for the corresponding workload requirements. After the okay by the architect, the analyst declares the set of NFRs in the integrated requirements specification as conflict-free.

5.8 Learning from Project Experience

In such a young discipline as software engineering, learning from project experience is crucial in order to optimize engineering in future projects. Concepts like the experience factory [BCR94b] ensure that for a specific project context, the optimal set of inputs (e.g., processes, techniques, methods, models and tools) is used. Our NFR methodology already uses experience-based artifacts, even though they are not organized in such a

formal way as an experience factory. In the future, quality attributes in quality models could be enhanced with a context vector that describes for which kinds of products and projects in a company the quality attribute seems to be suitable. With this information, the tailoring of the quality models (P1.2) could be supported and partially automated. This might make sense if very large quality models would be in scope for the projects.

We know from approaches like QIP [BR88] and the experience factory that analytical activities in the late phases of the project, i.e., learning from project experience, is essential for keeping the experience base up to date. Therefore, the NFR methodology also makes use of a learning activity:

After phase 2 of the NFR process, there may be important information in the project-specific artifact that should be incorporated into the experience-based artifacts. Besides general know-how that one wants to capture, the following information is of specific importance:

- New information emerged during phase 1: If there is new information in QM_InScope that is not in QM_Ref, the responsible requirements analyst should carefully think about whether the new information should be added to QM_Ref. This can, of course, also effect the reference checklists or the reference template. This information might be:

– A new QA that is inserted into QA_InScope during P1.2

– A new dependency that is captured in the dependency matrix during P1.3

- New information that has emerged during phase 2:

– New types of NFRs: During P2.1, the customer comes up with NFRs that do not constrain any $QA \in QA\_InScope$, but the NFR would constrain a new QA. This new QA would then be added to the QA_InScope.

– New dependencies: During the dependency analysis (P2.2) a dependency is found between two NFRs $n_1, n_2 \in NFR\_E$, but it holds for $q_1, q_2 \in QA\_InScope \mid (n_1, q_1) \in isOfType$ *and* $(n_2, q_2) \in isOfType$ that $(q_1, q_2) \notin influence$. This means that a dependency was found at the NFR level that is not reflected in the current dependency matrices. In this case, $(q_1, q_2) \in influence$ would be added to the QM_InScope.

One would therefore check whether the newly introduced QA or dependency is project-specific or whether it should be incorporated into QM_Ref. However, new information that emerged in phase 2 should not only be used for improving the experience-based artifacts, but should also impact the current NFR process. New NFRs make an additional iteration of P2.1 necessary, as the new QA was not in scope at the beginning

of P2.1 (see Section 6.1 on how to use the NFR process iteratively). New dependencies found in P2.2 make a new iteration of P2.2 necessary.

6 Increasing the Method's Efficiency

Having an effective method for eliciting and specifying a complete set of NFRs is important. Mechanisms for increasing the efficiency of the method can be a key enabler for its practical applicability. As can be seen from the description in Chapter 4, many parts in the NFR methodology are intentionally constructed in an algorithmic or pattern-based way so that they can be easily automated. [Ebe98] already stated the need for tool support: "We identified a big need for tools for management of NFRs." Furthermore, the NFR methodology foresees several places where focusing can be used in order to trade off results and effort. Both the tool support and the focusing shall make the NFR methodology more pragmatic and applicable to real-life projects. The first section in this chapter deals with the possibilities of focusing the effort, the second section with the tool support developed for this NFR methodology.

6.1 Focusing the Effort

[Boe81] already stated that the more detailed a guideline document is, the less the guidelines will be used. Therefore, if the NFR methodology uses a checklist or checklist questions incorporated into the tool-supported version, it is essential that one can set focus in order to reduce the number of questions. Focusing the effort has two goals: First, any effort that is less beneficial shall not be spent on the NFR methodology. Second, the NFR methodology shall enable the requirements engineer to trade off between spending additional effort to find more NFRs and saving the effort because the NFRs might not be that relevant. Comparing requirements engineering with risk management, the possibility of focusing the effort for elicitation is a means for adjusting the risk of missing critical NFRs. With regard to these two goals, the NFR methodology enables focusing in four areas:

- Focus by setting the quality scope: The $QA \in HLQA$ are prioritized and the set of QAs that shall be subject to elicitation is determined. The focus is set in P1.1 (prioritization of QAs) and P1.2 (selection of QAs). The effect of this focus is on all subsequent process activities.
- Focus by setting the functional scope: Horizontal and vertical scope selection is performed in P2.1. By this activity, only the most important functional items are the focus of elicitation or the NFRs are only elicited up to a certain level of abstraction. The effect of this kind of focusing is on P2.1.

Focus Area:	Goal: inherently cut less beneficial effort	Goal: enable trade-off between effort & completeness
Quality Scope	√	√
Functional Scope		√
Restriction of Dependencies	√	√
Relationship FR/SYS/NFR	√	

Table 10: Goals of the focus areas

- Focus by restricting the dependencies to be checked: By evaluating the possibly conflicting pairs in P1.3 and by restricting the check of NFRs to those that affect the same functional item/system component in P2.2, less effort is needed for the dependency checking. The decisions have an effect on P2.2.
- Implicit focusing by using the relationships between quality elements and functional conceptual elements / system components: This focus is inherent to the entire NFR methodology, as it is anchored in the metamodel for the methodology. It has an effect on almost all process activities, with the strongest one being on P2.1.

Some of the focusing can be steered by the requirements engineer during the process, while other focusing is inherently built into the method. Table 10 gives an overview of which focus areas are used for which goal.

As can be seen from the table, the requirements engineer can focus on three of the four focus areas. In the following, more details of these trade-off decisions are presented. Furthermore, details will be presented on how the NFR methodology can be performed in an iterative way by addressing certain quality attributes or functional elements in different iterations.

6.1.1 Trading off the Quality Scope

In this trade-off decision, the requirements analyst together with the customer determines which quality attributes are in scope. The quality scope can be affected in four areas:

1. Prioritizing the set HLQA: The customer determines the order of importance for all elements in HLQA. The quality focus is already influenced by the set of $QA \in HLQA$ that are chosen for prioritization. Furthermore, the order has an effect on the quality models that will be in scope later (QM_InScope).

2. Selecting the *QA* ∈ *HLQA* that are in scope for the project: Usually the customer decides on the number of *QA* ∈ *HLQA* that will be in scope. A typical number is two to three QAs. Another option would be to not determine the number of QAs in scope, but rather to take an iterative NFR elicitation approach, i.e., to start with the first one (highest priority QA), finish the NFR elicitation, and see whether there is enough time and effort left for the second QA, and so on.

3. Tailoring of the QM: In this activity, a focus is set by determining the relevant child QAs for the high-level *QA* ∈ *HLQA*. If a QA is considered not important, it is simply left out of the quality model. In case one is unsure whether a QA is important or not, we recommend including it in the quality model. We want to emphasize that the tailoring of the quality model could theoretically be used to set the focus for the next iteration by making a very small quality model, but this is not the intention of this step and may lead to the dangerous situation that some QA are forgotten for elicitation, i.e., NFR_E will become incomplete. This kind of adjustment (to take a subset of QAs from the quality model) can be performed in the next activity.

4. Selecting the *QA* ∈ *HLQA* for the next iteration. If the NFR methodology is performed in an iterative way (see also area 2: Selecting the *QA* ∈ *HLQA* that are in scope for the project), one can select at the beginning of P2.1 those quality attributes that shall be in scope for the current iteration. This gives the requirements engineer the chance to complete the NFR elicitation for one quality attribute rather than having a scattered set of NFRs for various quality attributes because the time is not sufficient for a complete elicitation.

We want to emphasize again that only the areas 1, 2, and 4 should be used intentionally to set the quality scope.

6.1.2 Trading off the Functional Scope

The main goal of this NFR methodology is to achieve a complete set of NFR_E. As the methodology is built on the relationship between quality attributes and functional conceptual elements and system components, respectively, it is dangerous to limit the functional scope for the elicitation. But there may be settings where a reduction of the functional scope may make sense, such as:

- There is not enough time to perform a complete NFR elicitation;
- Requirements are scheduled for various releases and the NFR for requirements of the first release should be elicited first;
- Some functional requirements are in a document where the components are already built. Therefore, the NFR elicitation shall only be performed for the new requirements;

- Some functional requirements are in a preliminary state and are therefore not mature enough to be input to the NFR process.

Therefore, the NFR methodology foresees the possibility to set the functional scope for the elicitation:

- Vertical scope selection: In this case, the scope is set by the level of abstraction that is the target of the NFR elicitation. The most common way to set the abstraction level is, for example, to set the UT_E as the target, but not to select the ST_E (which corresponds to the set of system functions) or the DI_E as a target for the NFR elicitation. We recommend always selecting SYS as the target.
- Horizontal scope selection: In this case, some functionalities and/or components are more important than others. The requirements analyst selects specific user tasks from UT_E and system components from SYS to be the target of the NFR elicitation.

A combination of horizontal and vertical scope selection is also possible. This functional scope selection can also be used when the NFR methodology is used iteratively. Then some functionalities or abstraction levels are the subject of the first iteration, whereas other functionalities or abstraction levels can be scheduled for future iterations.

6.1.3 Trading off the Dependencies

When focusing the effort with regard to the dependency checking, we distinguish two areas of focusing:

- Focus the dependency checks by only comparing NFRs where the QAs, the NFRs are type of, are marked as influencing each other in P1.3.
- Focusing dependencies on NFRs that constrain the same functional conceptual element/system component.

By reducing the number of comparisons to those where only the related QAs that influence each other are checked, the NFR methodology significantly reduces the number of comparisons:

Let

$$QA\_NFRexpressed=\{QA_i \in QA\__{InScope} \mid \exists\, n_j \in NFR\_E, (n_j, QA_i) \in isOfType\}$$

n=#elements of NFR_E

q=|QA_NFRexpressed|=number of QAs for which at least one elicited NFR is type of

r=number of relationships between elements of QA_NFRexpressed in dependency matrix

c=n/q = average number of NFR per QA

Then the number of comparisons (including intra-QA and inter-QA dependencies) NC_{all} needed to find dependencies without restriction to the ones where QAs are marked as influencing each other amounts to

$$\mathrm{NC}_{\mathrm{all}} = \sum_{\mathrm{i}=1}^{\mathrm{n-1}} \mathrm{i} = \frac{n*(n-1)}{2} = \frac{n^2-n}{2}$$

since a pair-wise comparison of all NFRs is needed. If we follow the algorithm that only compares NFRs where the QAs are related (see Section 5.7), the number of comparisons for intra-QA dependencies $NC_{rel\text{-}intra}$ amounts to:

$$\mathrm{NC}_{\mathrm{rel\text{-}intra}} = q*\sum_{i=1}^{c-1} i = \frac{q*c*(c-1)}{2} = \frac{qc^2-qc}{2}$$

as a pair-wise comparison for all NFRs expressed for each $QA_i \in$ QA_NFRexpressed needs to be performed. The number of comparisons for inter-QA dependencies $NC_{rel\text{-}inter}$ amounts to:

$$\mathrm{NC}_{\mathrm{rel\text{-}inter}} = r*c^2$$

as for each relationship, each NFR of the two influencing QAs need to be compared. So the total number of comparisons NC_{rel} is

$$\mathrm{NC}_{\mathrm{rel}} = \mathrm{NC}_{\mathrm{rel\text{-}intra}} + \mathrm{NC}_{\mathrm{rel\text{-}inter}} = \frac{qc^2-qc}{2} + r*c^2$$

If all QAs in QA_NFRexpressed were related, then

$$r = \sum_{i=1}^{q-1} i = \frac{q*(q-1)}{2}$$

and the total number of comparisons would be

$$\mathrm{NC}_{\mathrm{rel}} = \frac{qc^2-qc}{2} + r*c^2 = \frac{qc^2-qc}{2} + c^2*\frac{q*(q-1)}{2} =$$

$$\frac{c^2q^2 - cq}{2} = \frac{n^2 - n}{2} = \mathrm{NC}_{\mathrm{all}}$$

But usually the number r of related QAs in QA_NFRexpressed is smaller resulting in $NC_{rel}<NC_{all}$. Let us take, for example, the quite full dependency matrix (r=13, q=8) in Figure 35 and assume that c=3, i.e., on average 3 NFRs constrain one QA, which amounts to a total of 24 NFRs. Then instead of

$$\mathrm{NC}_{\mathrm{all}} = \frac{(3*8)^2 - 3*8}{2} = 276$$

comparisons only

$$\mathrm{NC}_{\mathrm{rel}} = \frac{8*9 - 8*3}{2} + 13*9 = 141$$

comparisons need to be performed. If we consider that a set of 24 NFRs is not very large, but still 141 comparisons would be needed, we see that we either need good tool support that brings the candidate pairs to the attention of the requirements analyst, or we need to further restrict the dependency checking. This can be done as already announced by focusing the dependency checking only on those NFRs that restrict the same functional object or system component. In the current example, let's assume that for the inter-QA dependencies always exactly one NFR of each $QA_i \in QA\_NFRexpressed$ relates to the same functional conceptual element or system component. This means that for each relationship r between QAs, we need to make 3 comparisons (each pair of NFRs that relate to the same object).

For the intra-QA dependencies, we assume that two out of the three NFRs relate to the same functional conceptual element or system component. This means we have to make one comparison for each $QA \in QA\_NFRexpressed$. Thus, the number of comparisons would amount to

$$\mathrm{NC}_{\text{rel-restricted}} = 13*3 + 8 = 47\ .$$

comparisons. This would be a realistic number to be checked. In one of the case studies, where we created complete dependency matrices for real-life systems for a big company, we had a set of q=59 QAs in the models and about 564 relationships captured in the dependency matrices. If we assume that we have on average 10 NFRs per QA and for every QA one NFR was expressed (i.e., $QA\_NFRexpressed=QA\_InScope$), then we see the huge number of comparisons that would be needed to check the 590 NFRs:

$$NC_{all} = 173755; NC_{rel} = 59055 .$$

If we assume a situation where for the inter-QA dependencies from the 10 NFRs per QA, always one NFR relates to the same functional or system object (see previous example) and for the intra-QA dependencies an average of two pairs of NFRs relate to the same object, the number of comparisons would amount to

$$NC_{rel\text{-}restricted} = 5758 .$$

These figures show that there is an urgent need for research in the area of handling NFR dependencies by future work.

6.1.4 Performing the NFR Methodology Iteratively

As was already mentioned above, the NFR methodology is designed in such a way that especially the processes P2.1 and P2.2 can be used in an iterative way. This means that one can focus on certain quality attributes and functional conceptual elements or system components in a first iteration and add further elements in future iterations. The algorithm was created in a way that supports this iteration. One aspect is the marking of objects as compared or "done" in the comparison matrix. This is an essential feature in order to allow an efficient iterative approach, as one does not want to ask for an NFR again if the customer already thought about it. Figure 41 shows a comparison matrix after a first iteration of an example project. We see that:

- The elicitation of Data QAs was out of focus for the complete project (black marker). This was an implicit decision when creating the quali-

Items to be compared		User Task QAs					System Task QAs			System QAs			Data QAs		
		QA_1	…	QA_n			QA_{n+3}	…	QA_m	QA_{s+1}	…	QA_p	QA_{p+1}	…	QA_t
					Child System Task QAs										
					QA_{n+1}	QA_{n+2}									
User Task 1		done	done												
	System Task 1						done	done							
	…						done	done							
	System Task n						done	done							
User Task 2															
User Task …		done	done												
User Task n		done	done												
System Task n+1							done	done							
…							done	done							
System Task m															
System 1										done	done	done			
…										done	done	done			
System n										done	done	done			
Data Item 1															
…															
Data Item n															

Figure 41: Comparison matrix after first iteration

ty models: No QAs of the type Data were included in the model.

- During P2.1, the quality scope was set. The following QAs were excluded for the first iteration:

– User Task QA QA_n and corresponding child QAs QA_{n+1} and QA_{n+2} were not selected for the first iteration.

– System Task QA QA_m was not selected for the first iteration.

- During P2.1, the functional scope was set. All system components were selected for the first iteration. The following functional elements were excluded for the first iteration:

– User Task ut_2 was not selected for the first iteration.

– System Task st_m was not selected for the first iteration.

Thus, in Figure 41, the cells with the dotted background represent the items that were out of scope for the first iteration.

So if one were to select all remaining QAs and functional elements for the second iteration of P2.1, the instance of the algorithm for the second iteration would look as illustrated in Figure 42. After this second iteration, all open elements in the comparison matrix are marked as done. The NFR elicitation is complete. Checking for dependencies (P2.2) can and should also be restricted to the newly found NFRs.

For each User Task QA $utq \in \{QA_1, \ldots, QA_{n-1}\}$:
    For ut_2:
        Ask for NFR: IF (NFR=necessary) THEN specify NFR,
        ELSE -
        Mark ut_2xutq as done
For QA_n:
    For each $ut \in \{ut_1, \ldots, ut_n\}$:
        Ask for NFR: IF (NFR=necessary) THEN specify NFR,
        ELSE -
        Mark $utxQA_n$ as done
    For each $stq \in \{QA_{n+1}, QA_{n+2}\}$:
        For each $st \in \{st_1, \ldots, st_n\}$:
            Refine NFR: IF (NFR=necessary) THEN specify NFR,
            ELSE -
            Mark *stxstq* as done
        For each $st \in \{st_{n+1}, \ldots, st_m\}$::
            Ask for NFR: IF (NFR=necessary) THEN specify NFR,
            ELSE -
            Mark *stxstq* as done
For each System Task QA $stq \in \{QA_{n+3}, \ldots, QA_{m-1}\}$:
    For System Task st_m:
        Ask for NFR: IF (NFR=necessary) THEN specify NFR,
        ELSE -
        Mark st_mxstq as done
For System Task QA QA_m:
    For each $st \in \{st_1, \ldots, st_m\}$:
        Ask for NFR: IF (NFR=necessary) THEN specify NFR,
        ELSE -
        Mark stxQA_m as done

Figure 42: Exemplified instance of NFR algorithm for second iteration

6.1.5 Change Management View

The capability of the NFR process to deal with iterations can also be used for another important aspect in requirements engineering. In this section, we discuss the change management view on focusing the activities. As Bob Dylan already said, "there is nothing so stable as change". This is especially true in requirements engineering [VDG08]. Therefore, if functional requirements or the system's physical architecture change, the NFR methodology must be capable efficiently handling this change: A change is usually:

- an addition of an element,

- a modification of an element, or
- a deletion of an element.

An addition of an element leads to a new row being added in the comparison matrix. A modification of an element leads to the row of elements in the matrix belonging to the changed element being emptied and to all NFRs that relate to this element being marked as "suspect" in the integrated requirements specification. A deletion of an element leads to a deletion of the row in the comparison matrix and all NFRs that relate to this element are marked as "deletion candidates" in the integrated requirements specification. Then, the NFR methodology can be applied again and will act very similar to just having a new iteration. The only change is that the NFRs in the integrated requirements specifications that are marked as "suspect" or "deletion candidate" must be checked for validity. For the NFRs marked "suspect", a good time to check the validity is when the algorithm arrives at the corresponding advice to state QAs for the functional object or system component. Then, besides thinking about the necessity of new NFRs, one should check whether the "suspect" NFR is still needed or needs to be modified. We recommend doing the validity check for the "deletion candidate" NFRs after P2.1. Most likely, the NFRs will also be deleted, as the functional conceptual element or system component they belong to was deleted. But this is not always the case, since an NFR can also be related to a deleted object, but as it was classified as over spanning NFR, it must remain in the integrated requirements specification.

6.2 Tool Support

Existing NFR Tools

Only few requirements engineering tools exist that specialize on NFRs. [BSD+07] published work on their tool ElicitO which claims to support NFR elicitation by making use of a functional and quality ontology. But the tool has two major drawbacks: neither is there an elicitation guidance in the sense of an elicitation algorithm as in this NFR methodology, nor is the tool adequate for complete functional modeling, unless the functional requirements should be stored in an ontology like fashion. This is due to the fact that the tool does not differentiate or support functional objects besides functional domain activities. The IVENA tool from Sophist Group in Germany is basically a large, structured database of NFRs that were specified in previous projects. The NFRs are structured according to a quality attribute hierarchy similar to the one in the quality models of this NFR methodology. The problem with this kind of tool is twofold. First, the database soon becomes to large to find relevant NFRs. Second, there is the risk that NFRs that did not appear in previous projects are missed during NFR elicitation.

Tool-support in the NFR-methodology

Some parts in the NFR methodology are intentionally constructed in an algorithmic or pattern-based way so that they can be easily automated. Figure 43 gives an overview of the process activities for which prototypical tool support is available as part of the NFR methodology. First, there is a Checklist Generation Tool to automate the checklist part of process activity P1.4. This tool was developed as a Visual Basic for Application Plugin in Microsoft Office Viso. It takes as input a quality model that is element of QM_InScope (for example the one in Figure 33) created in Microsoft Visio and produces the checklist for elicitation as a Microsoft Word document. We decided to implement the Checklist Generation Tool based on Microsoft Office products, as the Checklist Generation Tool shall support manual elicitation and the Microsoft Office tool suite is commonly used. So the Checklist Generation Tool can be used by almost everybody. The second tool support aims at the elicitation and specification activity (process activity P2.1). This Elicitation Guide Tool is an extension of IBM Telelogic Doors using the Doors extension language (DXL).

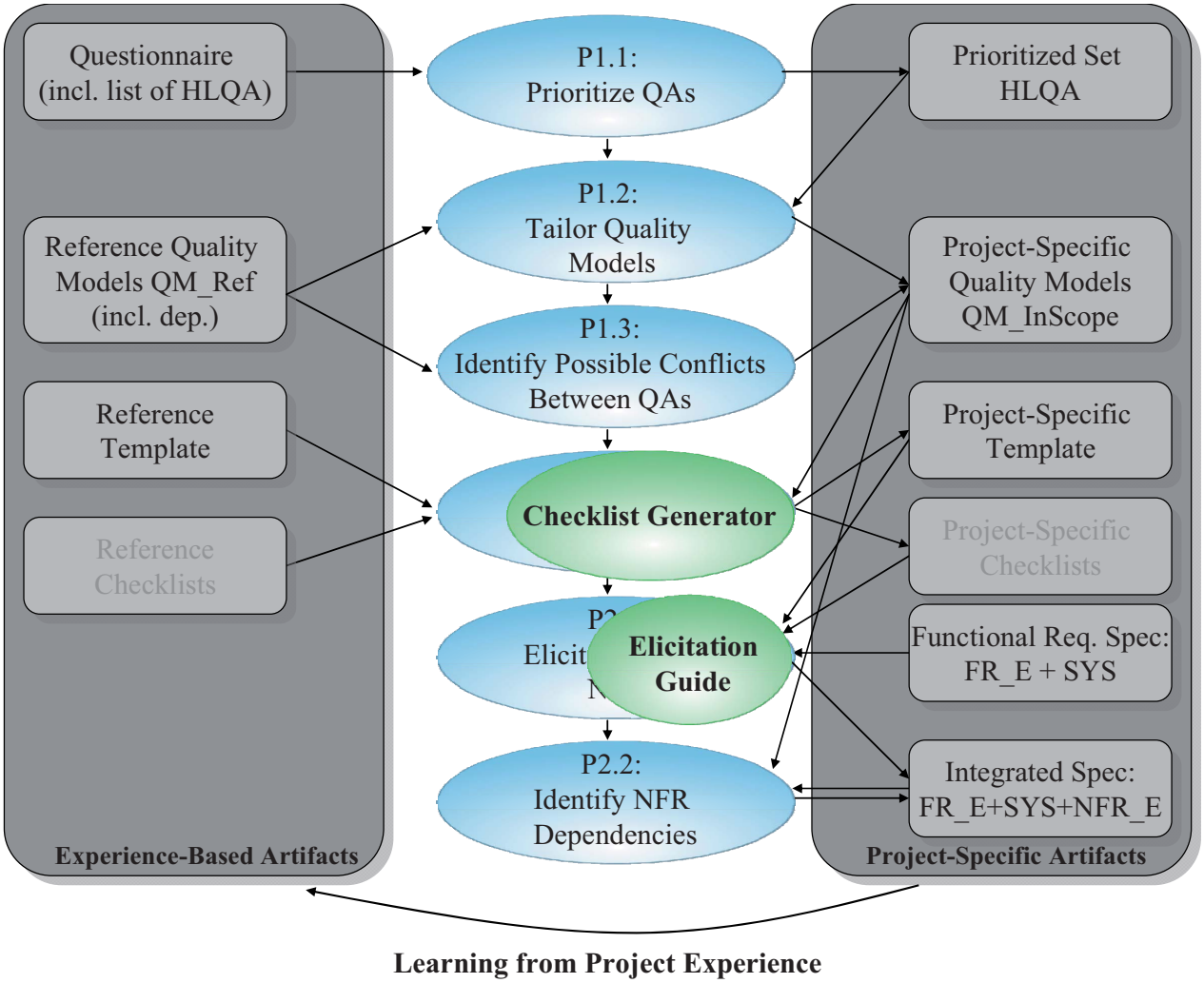


Figure 43: Tool support for the NFR process

The tool has the elicitation algorithm built in and asks directly for NFRs, showing the related functional conceptual element or system component, respectively. The tool has the comparison matrix (see Figure 29) built in and therefore knows which comparisons have already been executed. This prototype was developed as a DXL Plugin, since Doors is a widespread requirements management tool. Both tools interact via a defined interface. The output of the Checklist Generation Tool can be used as input for the Elicitation Guide Tool.

6.2.1 The Checklist Generation Tool

The Checklist Generation Tool automatically generates the checklists that are used for the NFR elicitation in process activity P2.1. Figure 44 illustrates the basic design of the tool showing the input and output of the tool as well as the logical internal components.

Using the Tool

The Checklist Generation Tool takes the quality model QM_InScope$_x$ as input. Figure 45 shows the startup screen of the Checklist Generation Tool.

One has to first select the root of the out-tree. This is done by choosing the parent QA. Then the model is checked to see whether it satisfies the constraints for being an out-tree.

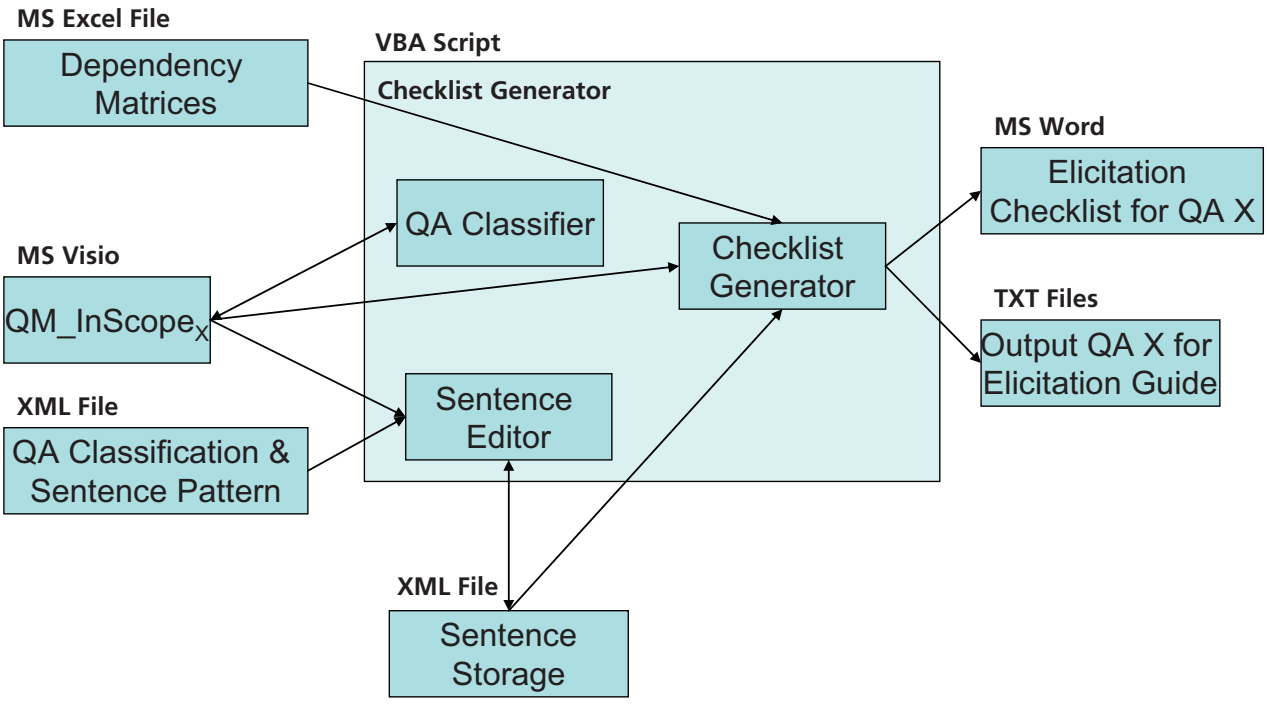


Figure 44: Basic design of the Checklist Generation Tool

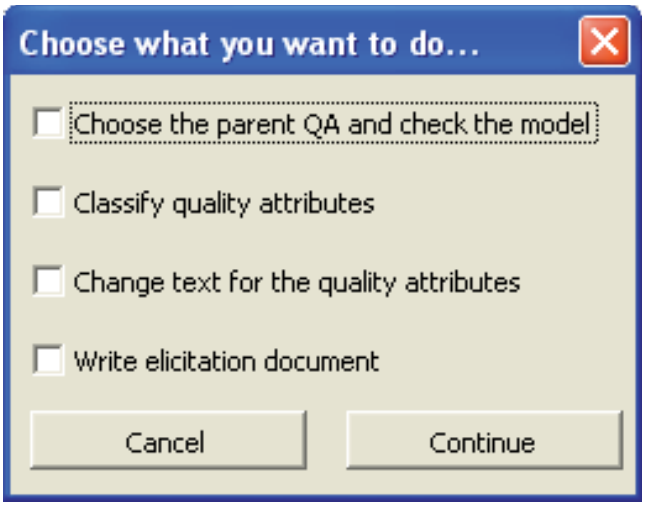


Figure 45: Main-Screen of Checklist Generation Tool

If this is done, one can select from the next three possibilities:

- Classify quality attributes: This is a support functionality for process activity P1.2. If the QAs in $QM\_InScope_X$ are not yet classified according to the scheme User Task, System Task, System, Data and Structural QA, the requirements analyst can now classify the attributes by choosing this menu item. This functionality is provided by the internal component "QA Classifier".
- Change text for the quality attributes: This is support functionality for process activity P1.4. Based on the sentence patterns stored in the XML File "QA Classification & Sentence Pattern", the internal component "sentence editor" will generate the advice for the checklist by applying the sentence patterns to all quality attributes in $QM\_InScope_X$. In some cases, one wants to edit the sentences for the checklist in order to make the advice more precise or meaningful. E.g., based on the sentence pattern for System QA, the Checklist Generator would transform the System QA "Throughput" into the sentence: "For each system part, please specify the throughput NFRs." One might want to edit this sentence to "For each system part that is a network, please specify the throughput NFRs." The sentence editor stores this information in the XML File "Sentence Storage". This means the edited sentences are automatically kept for the future. If one changes the $QM\_InScope_X$ at a later point in time, but leaves the QA "Throughput" unchanged, the internal Component Checklist Generator will use the changed sentence from the file "Sentence Storage".
- Write elicitation document: This is the actual core of the checklist generator, which creates the elicitation checklist as a Word file and as text files for the Elicitation Guide Tool. The checklist generator takes

the QM_InScope$_X$ and inserts an advice for each QA into the Word document. It therefore makes use of the sentences stored in the sentence storage file.

Additional Features

The tool has two additional features with regard to the place for specifying the elicited NFRs and the dependency checks between the NFRs:

- Specification of the documentation path for NFRs: The sentence editor allows for specifying the documentation path for an NFR. As the place to document the NFR is given by the requirements template and the classification of the QA the NFR is type of, one can add the documentation path to the elicitation checklist. This is support for the requirements analyst, as it tells him or her where to document the elicited NFR.
- Integration of dependency check during elicitation: The internal component "checklist generator" has an additional feature: One can select whether one wants to incorporate the information on possible dependencies into the checklist. If one selects to use this feature, the checklist generator takes the information from the dependency matrices (xls files) and generates this information into the Word file. For this, it uses the following sentence pattern:

– Typical intra-quality model dependencies. Please keep in mind the following relationship(s):
    - The *Name of QA* NFR typically influences *Name of conflicting QA* quality attribute(s), and viceversa.
    - The *Name of QA* NFR is identical to *Name of identical QA*quality attribute(s).
    - The *Name of QA* NFR typically influences *Name of conflicting QA*quality attribute(s).
– Typical inter-quality model dependencies. Please keep in mind the following relationship(s):
    - The *Name of QA* NFR typically influences *Name of conflicting QA*, and viceversa.
    - The *Name of QA* NFR typically is identical to *Name of identical QA*.
    - The *Name of QA* NFR typically influences *Name of conflicting QA*.

Each bullet is only introduced if dependencies of that type exist in the xls files to make the elicitation checklist as small as possible. The bullets are directly inserted behind the text of the advice (which can include the information on available metrics).

We recommend to carefully trade off whether this feature should be used for a project. On the one hand, it integrates P2.1 and P2.2. into a joint activity. On the other hand, it makes the checklist larger. Furthermore, the dependencies are checked directly after elicitation of that

Elicitation of Efficiency NFRs

…

1. Elicitation of Time Behavior NFRs

1.1 Elicitation of Usage Time NFRs *(UC-Diagram→@Use Case: Usage Time)*

For each use case, please specify the Usage Time

- Please use the available metrics: seconds

1.1.1 Elicitation of Response Time NFRs *(System Function Description→@System Function:Response Time)*

For all Use Cases: check if this Use Case has one or more NFRs of the type Usage Time: If so, please refine these NFRs by specifying Response Time NFRs for each system function that is used in this Use Case.

For all other system functions: please specify the Response Time

- Please use the available metrics: seconds

2 Elicitation of Accuracy NFRs

2.1 Elicitation of Precision of Calculation NFRs *(General NFR → Efficiency→Accuracy→Precision of Calculation)*

For each system function, please specify the Precision of Calculation

- Please use the available metrics: % of Error in Rounding

2.2 Elicitation of Precision of Data Storage NFRs *(Data Model→@Data Item: Precision of Data Storage)*

For each data item, please specify the Precision of Data Storage

- Please use the available metrics: #digits after comma
  - Typical intra-quality model dependencies. Please keep in mind the following relationship(s):
    - The Precision of Data Storage NFRs typically influences Capacity of Memory, Capacity of Secondary Storage quality attribute(s), and viceversa.

…

Figure 46: Excerpt of reference checklist for $QM\_Ref_{Efficiency}$ generated with additional features

type. For a bigger project, it might be more reasonable to wait until NFR_E has been completely elicited. Therefore, we consider this feature more useful for smaller projects.

To provide an idea of the resulting checklist, Figure 46 shows an excerpt of a checklist that was created by using these two additional features. One can see behind the captions for each advice the documentation places in italics. Furthermore, for the QA "Precision of Data Storage", the dependencies according to Figure 35 were inserted.

Limitations

Even though the tool already provides good support for checklist generation, it also has some limitations:

- The concept of means is not supported in the current version of the VBA Plugin.
- The order of advices is not correct according to the algorithm; it is aligned according to the structure of the quality model.

Those limitations could be addressed in future work.

6.2.2 The Elicitation Guide Tool

If the Elicitation Guide Tool is used, no checklist needs to be created or generated. As depicted in Figure 43, the Elicitation Guide Tool supports P2.1. More specifically, the Elicitation Guide Tool automates asking the user the elicitation questions and automatically documents the NFRs in the correct place. Therefore, in theory, no requirements analyst needs to participate in the NFR elicitation session. For practical reasons (acceptance of the customer), it might be more appropriate at the beginning that a requirements analyst uses the tool together with the customer or end user who has the domain expertise to issue the NFRs. Figure 47 illustrates the basic design of the Elicitation Guide Tool.

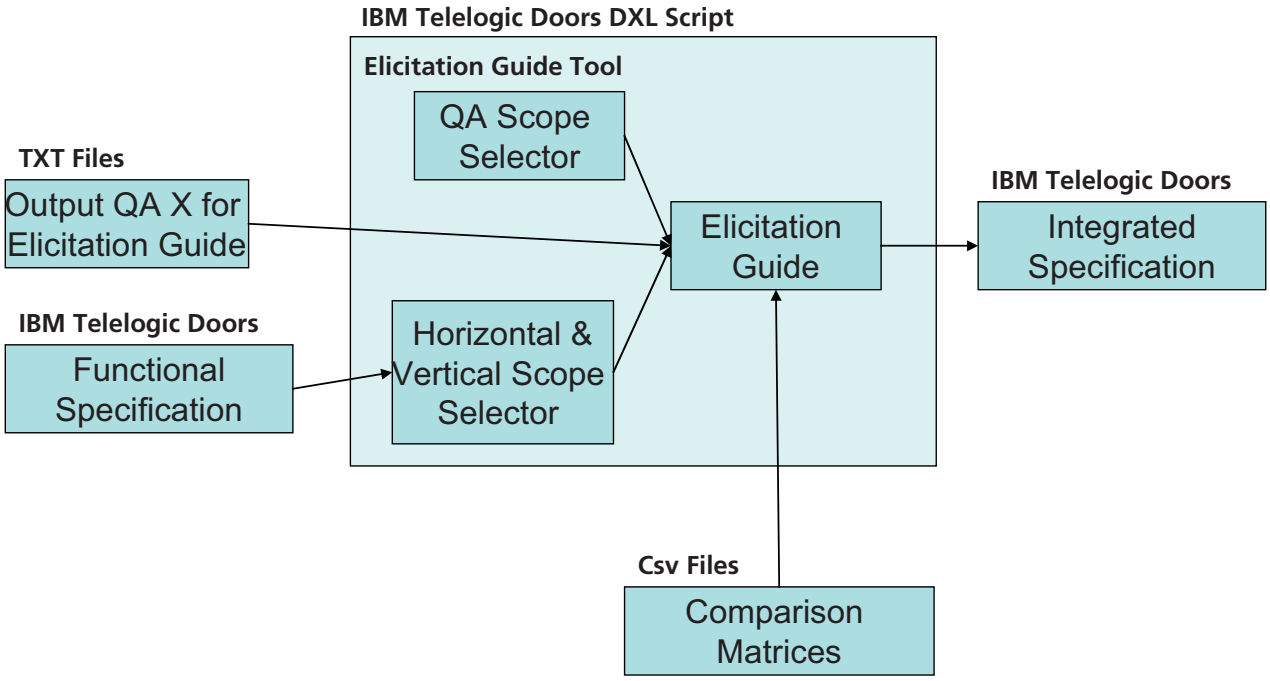


Figure 47: Basic design of the Elicitation Guide Tool

The Elicitation Guide Tool is integrated as DXL-Script in Telelogic Doors. For all graphical elements such as UC diagrams, class diagrams, or component diagrams, the Doors Analyst extension is used to integrate graphics in Doors. The Elicitation Guide Tool is typically run after the functional requirements have been entered in Doors. The tool can deal with the following structure of the functional requirements:

- User tasks: UCs in UC diagrams
- System functions: steps in UC descriptions in tabular UCs
- Data objects: classes in class diagrams
- Subsystems: components in component diagrams

In addition to the functional requirements, the tool takes the "Output QA X for Elicitation Guide" txt file, which is the output of the checklist generator (see Figure 44), as input.

Using the Tool

The tool starts with the start screen depicted in Figure 48. When using the tool for the first time in a project, typically the quality and the horizontal and vertical scope must be selected. Scope selection is performed as described in Section 6.1. Figure 49 shows the windows for the scope selection.

After the scope selection, the tool starts the elicitation algorithm by asking the user the elicitation questions while bringing the corresponding functional conceptual element or subsystem to the attention of the user. In this sense, the tool is much more concrete than the generated checklist. For example, the checklist questions typically ask "For all UCs: check if this UC has one or more NFRs of the type Usage Time". The Elicitation Guide Tool asks, "For this Use Case, please specify the Usage Time NFRs", showing the current UC. Figure 50 shows a screenshot for the

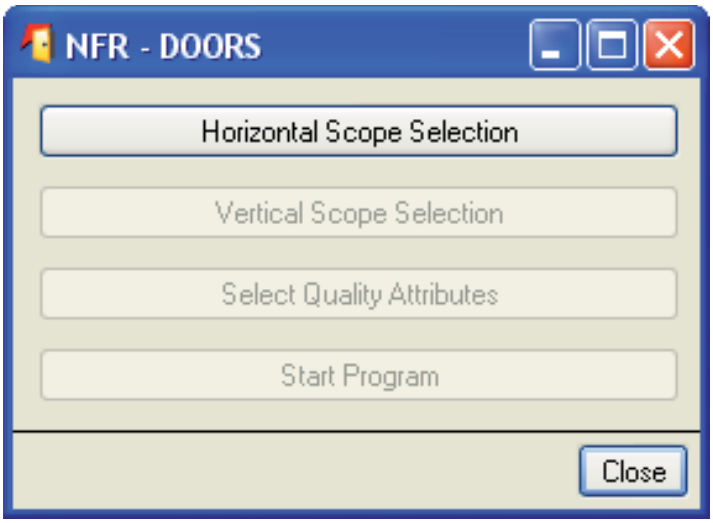


Figure 48: Start screen of the Elicitation Guide Tool

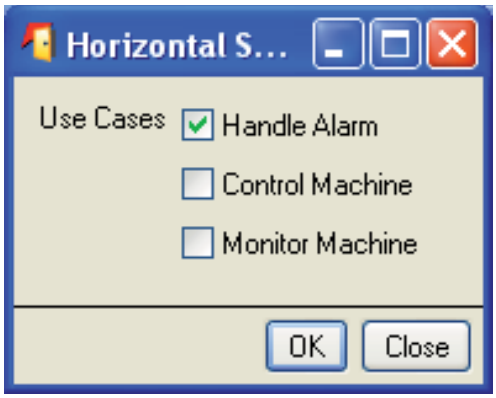


Figure 49: Horizontal scope selection in the Elicitation Guide Tool

elicitation. In this case, the system asks for refined System Task NFRs. Therefore, it automatically presents the User Task NFR that is to be refined (third row). This feature speeds up the elicitation process, as one does not have to look in the specification for the corresponding functional conceptual elements, subsystems, or related NFRs. The user can now select whether he or she wants to

- skip this question (using the "Next" button),
- state an NFR for this functional conceptual element or (sub-) system (using the text field and pressing the "Save NFR" button),
- state that no NFR exists for this question (using the "No NFR" button).

The tool stores all this information. NFRs are directly stored in the Doors database integrated with the functional requirements. The information that no NFR exists or that the question was skipped is stored in the comparison matrices as CSV files. Therefore, if the tool is run again, questions are only asked for skipped questions or items that are new in scope. This is beneficial for the iterative NFR elicitation and specification (see Section 6.1.4).

Additional Features

An overview of how complete the NFR elicitation was can be obtained by looking at the comparison matrices where the comparisons were stored.

Furthermore, the tool automatically links related NFRs as well as NFRs to related functional conceptual elements or subsystems. This is done by using the link technology provided by Doors.

Limitations

The tool has some limitations:

- All system function invocations must be located in the UCs. The tool currently does not support stand-alone system function definitions. Therefore, the NFRs for system functions can only be documented in the UC description.

- The tool cannot distinguish between system and user actions in the UC flow of events. Therefore, the user must manually skip these steps by pressing the "Next" button.
- The functional requirements must be documented in the structure as presented above. The identification of the functional conceptual elements and subsystems is hard coded and cannot be changed without editing the DXL source code.

For full support of the NFR methodology, the tool needs to be extended to deal with system function descriptions apart from the UC descriptions.

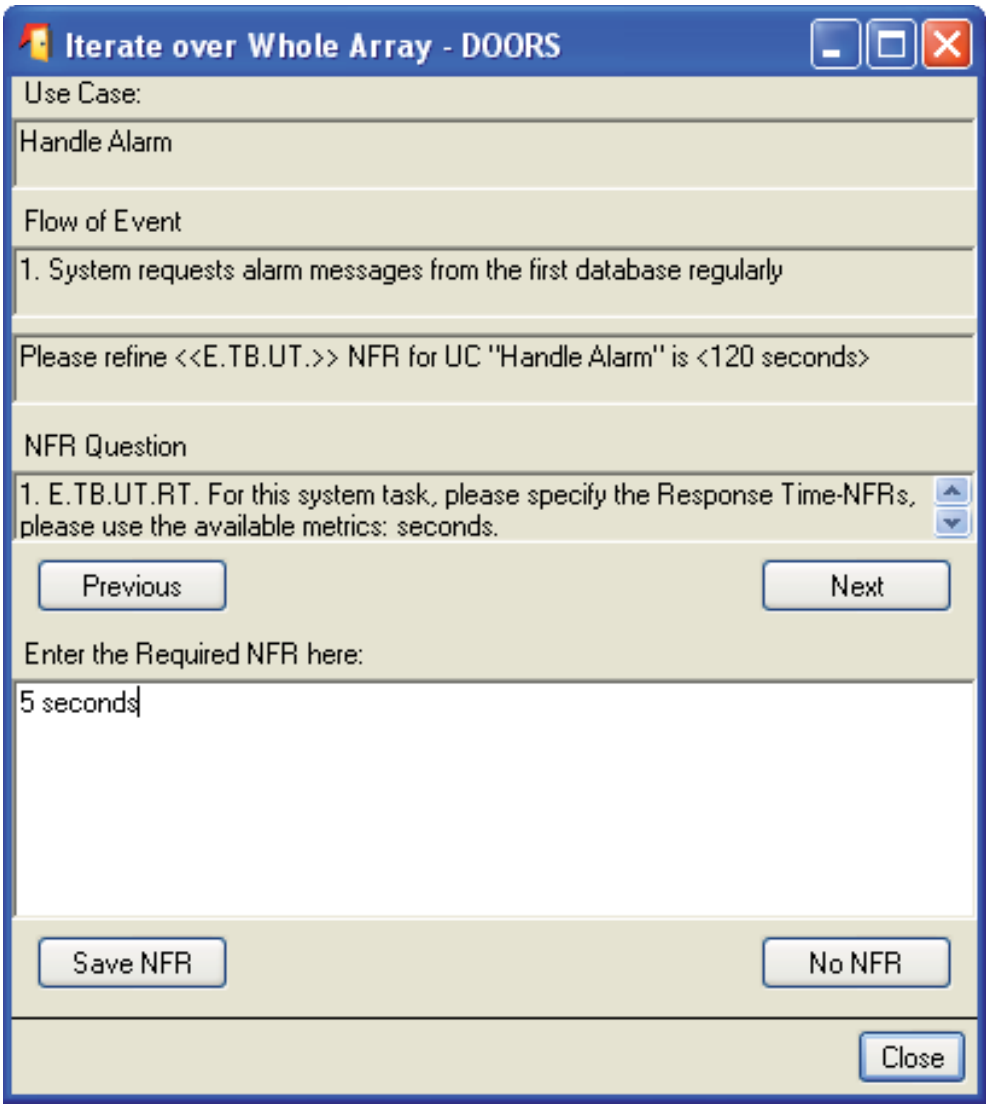


Figure 50: Elicitation support in the Elicitation Guide Tool

7 Validation

This chapter describes empirical results [Rom08] of the successful application and introduction of the NFR methodology in eight different mainly industrial project contexts. In each of the eight projects, different aspects of the NFR methodology were introduced, applied, and evaluated. The NFR methodology was evaluated according to the three hypotheses H1 – Feasibility, H2 – Completeness, and H3 – Effort.

Summary of Results

In general, the case studies revealed very positive results. The collected data strongly support hypothesis H1 – Feasibility: "The elements of the NFR methodology are feasible, i.e., the artifacts can be created for real-life examples and the process activities can be applied by averagely trained personnel." The results of the feasibility evaluation are very positive: Except for process activity P2.2, which was rated as too time-consuming for large projects, all parts of the NFR methodology are absolutely feasible from the viewpoint of the evaluated results produced as well as from the viewpoint of subjective opinions that were collected.

The collected data strongly support hypothesis H2 – Completeness: "The method results in a (more) complete set of NFRs. About 20% more critical NFRs are identified compared to the state of the practice." The percentage of newly identified NFRs in the case studies range from over 100% to over 600%. The case studies show that after application of the NFR method, no new NFRs emerged in the subsequent software development phases.

The collected data support hypothesis H3 – Effort: "The estimated rework effort in the subsequent project or platform development phases is reduced: The estimated saved rework effort for found NFRs is at least twice the effort spent on systematically eliciting the NFRs." The effort that needs to be spent on performing the NFR methodology is completely justifiable, especially if compared to the effort that can be saved in the subsequent phases by knowing the NFRs in advance. ROI values of 2 and 17 were achieved.

As a positive side effect, we observed that the NFR methodology can produce a set of NFRs that is completely measurable, and we learned that the method can reveal conflicts in the set of elicited NFRs.

Remainder of this Chapter

In Section 7.1, we introduce the setup of the validation. In Section 7.2, we present an overview of the case study contexts. In Section 7.3. the hypotheses from Section 1.2 are refined by using the GQM approach. In Section 7.4, we present the data and results from the case studies that are eventually summarized and discussed in Section 7.5.

7.1 Introduction

When selecting the appropriate means for validating the NFR methodology, we traded off the benefits and drawbacks of performing industrial case studies vs. controlled experiments. There were two main reasons why we chose industrial case studies as a means of validation rather than performing a controlled experiment:

1. In order to judge the impact of NFRs on the architecture, one needs reasonably large systems. Therefore, real-life case studies are more suitable, as it is not feasible to create such a specification and architecture for a controlled experiment.

2. In a controlled experiment, where one group uses this NFR methodology and another group uses another state-of-the-practice or state-of-the-art methodology, one has to ensure that the persons who are interviewed during the NFR elicitation process have exactly the same NFR know-how about the system. It is not possible to replicate this know-how for all persons without bringing the NFRs to the direct attention of the interviewed person. This means the person would either remember all NFRs that were recently injected, or they would have forgotten essential NFRs, if you injected them with a time delay of, for example, six months to avoid the remembering effect. So one would not be able to get reliable results on the differences of the methods.

We had the opportunity to validate different aspects of this NFR methodology in eight real-life case studies. As most of the case studies were performed in the context of industrial projects, we were not completely free to set up perfect case studies and control all parameters. Rather than that, the project constraints of our project partners imposed some limitations, e.g., with regard to data collection possibilities. Furthermore, it was not possible to use two state-of-the-art NFR methodologies one after another (e.g., MOQARE or a goal based approach first and then this NFR methodology to show that more significant NFR are found) due to time and effort restrictions on the part of the project partners. Therefore, the focus of the eight case studies was on evaluating feasibility, characterizing the effort, and comparing this NFR methodology to state-of-the-practice NFR elicitation and specification procedures with regard to completeness. We will discuss the implications of this situation in Section 7.5.3.

7.2 Case Study Contexts

The NFR methodology was applied in eight different contexts. Table 11 gives an overview of the case studies. We characterize the differences between the case studies by describing:

- The type of project: The type of project influences the possibilities for carrying out specific parts of the method.
- The size of the project: The size of the project gives an insight into the applicability of the method for systems of various scales.
- NFRs were already part of the existing system specification: When NFRs were already part of the specification and the NFR elicitation part of this NFR methodology was carried out, it was possible to evaluate whether new NFRs were found (i.e., applying the NFR methodology results in a more complete set of NFRs).
- The type of system: The NFR methodology is primarily designed for interactive systems. Therefore, it is interesting to observe whether it can also be applied to reactive embedded systems.
- Typical IS specification: If the existing functional specification is a typical interactive system specification, the NFR methodology should be

Project name	ITEA Empress project	Ricoh	Geographical information system	Large information system in financial domain	MBTech Group	Large information system solution provider	EU project "Emerge"	Large embedded system provider
Project abbreviation	Empress	Ricoh	GIS	FIN	MBTech	SOL	EMERGE	ES
Project type	industry in public project	industry	industry	industry	industry	industry	organizations in public project	industrial
Project size	medium	medium	small	large	small	large	medium-large	large
NFRs were part of the existing specification	yes	yes	no	yes	N/A	yes	no	yes
Type of system	Interactive System	Interactive System	Interactive System	Interactive System	Embedded System	Interactive System	Interactive System	Embedded System
Typical IS specification	yes	yes	yes	yes	no	yes	yes	partially
Quality models in scope	reliability, efficiency, maintainability	efficiency	security	efficiency, maintainability, reliability	efficiency, portability, reliability	efficiency, reliability, security	efficiency, usability, reliability, safety, security	reliability, maintainability, usability, robustness
NFRs were elicited for project	yes	yes	yes (but resulting NFRs not accessible for author)	yes	yes (but resulting NFRs not accessible for author)	no	yes	no
Person applying method	author together with other Fraunhofer staff	author together with other Fraunhofer staff	other Fraunhofer staff	author together with other Fraunhofer staff	other Fraunhofer staff and industrial partner	author together with other Fraunhofer staff and industry partner	other Fraunhofer staff	author together with other Fraunhofer staff
Case study specificities and method deviations	early version of method: no data objects	early version of method: no data objects	early version of method: no data objects	early version of method: no data objects	method was used for HW/SW system	technology transfer; method adaptation for business processes		technology transfer
More Information on case study available	[DKK+05]	[DKK+05]	[DKK+05]	confidential	[ADB+08]	confidential	confidential	confidential

Table 11: Overview of the eight case studies

easily applicable.

- Quality models in scope: The case studies had different quality attributes in scope.
- NFRs were elicited for project: This information classifies the case studies into those where we had an insight into the actual NFR elicitation and those where we had no possibility to evaluate the NFR elicitation performance.
- Person applying method: The person steering the NFR methodology can influence the validity of the case studies (see also threats to validity).
- Case study specificities and method deviations: The eight case studies were performed over time with slightly different versions of the method.
- More information on case studies available: Due to space and confidentiality reasons, the eight case studies cannot be explained in detail in this thesis. This row shows whether and where interested readers can obtain more information.

Project Type

Six out of the eight projects were industrial projects where the industrial partners asked Fraunhofer IESE to either apply the method for an industrial product or to transfer the technology to their company. Two projects were public projects where companies or public organizations served as stakeholders interested in a product to be built. In Empress, the product to be built was a fictive one (which was built as a prototype). In EMERGE, the project intends to built a prototype for a new, commerically available product. Therefore, in all case studies, industrial partners were involved as stakeholders who were asked for the quality attributes and non-functional requirements.

Project Size

The eight case studies varied in system size and therefore also in the size of the functional specification that was taken as input. Two systems (GIS and MBTech) can be characterized as rather small systems offering little functionality. Three case studies dealt with specifications of large systems (FIN, SOL and ES). EMERGE, as a medium to large system, had about 20 UCs, 40 system functions, and one system components as the system was treated as a black box. Empress elicited NFRs for 13 UCs and 5 system components. In the Ricoh case study, 27 UCs and 17 system functions were input to the method. In the FIN case study, there were more than 35 system functions with strong dependencies and complex data structures.

NFRs were Part of Existing Specifications

In five out of the eight case studies, NFRs were part of the existing specification. In the MBTech case study, we have no information. In GIS and EMERGE, the specification was created from scratch using the NFR methodology as the first mechanism for eliciting NFRs. Therefore, no baseline data is available to compare whether the NFR methodology revealed NFRs that would have not been found without the method.

Type of System and Typical IS Specification

Six out of the eight case studies were typical information systems exhibiting a typical textual representation of functionality similar to the one described in Section 3.1.1. MBTech was the first case study from the Embedded Systems domain that was a totally different kind of system. The system is a generator that contains not only software, but also hardware and mechanical elements. This implies the question of the feasibility of the NFR methodology for such kinds of systems.

Quality Models in Scope

Due to the differences in product and project objectives, the eight case studies had different quality attributes in scope. Figure 51 shows the relevance of quality attributes that appeared more than once in the case studies. This should not be interpreted as a statement of relevance for projects in general. This distribution shows that the NFR methodology is capable of dealing with these five different qualities. Additionally, the ES project dealt with the quality robustness, EMERGE with safety, and MBTech with portability. In FIN, the quality attribute maintainability was only treated with respect to the sub-attribute analyzability and the quality attribute reliability was only treated with respect to the sub-attribute recoverability.

NFRs were Elicited for Project

As already mentioned, the different case studies had different objectives: The SOL and ES case studies had the goal of creating quality models and doing a technology transfer. Therefore, no NFR elicitation (carrying out process activity P2.1, see Figure 32) was performed for these projects. In

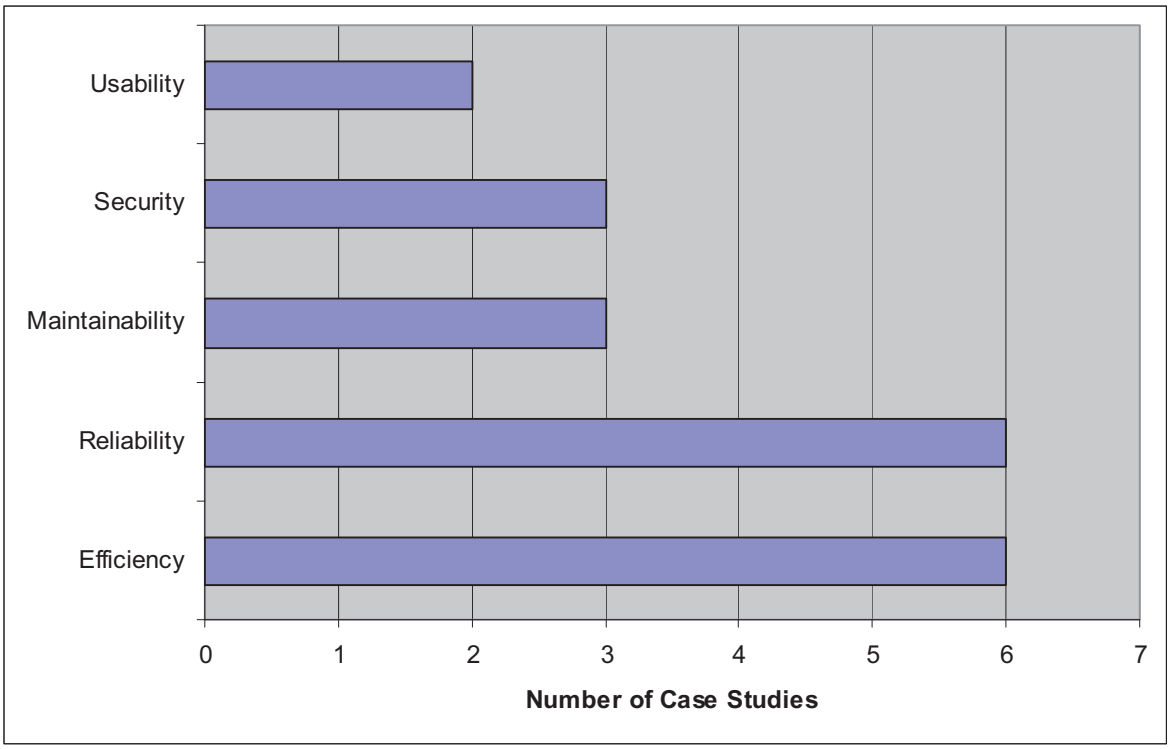


Figure 51: Quality attribute occurrence in case studies

the case of the SOL case study, the NFR elicitation process was performed in the form of role playing for a typical product to investigate the feasibility of the approach for the company. In all other six case studies, the actual NFR elicitation was applied for a concrete product.

Person Applying the Method

As the NFR methodology expert as a human being is part of the (at least non-tool-supported version of the) NFR methodology, this person can have a strong influence on the method's performance. The author of this thesis participated as an NFR methodology expert in four of the eight case studies to get direct feedback on the methodology's benefits and shortcomings. In these applications, other Fraunhofer staff always participated in order to get a more unbiased reflection on the method's performance. In three case studies (EMERGE, GIS, and MBTech), the case study was performed completely without the participation of the author, but by other Fraunhofer staff who were experienced in using the method. In the case of the MBTech case study, industrial persons did the NFR elicitation (process activity 2.1) by themselves.

Case Study Specificities and Method Deviations

Each case study has its own specificities with regard to the domain of the system as well as with regard to the way the NFR methodology was performed. In Section 1.3 we outlined that there were early and late case studies. The early case studies FIN, GIS, Ricoh, and Empress used an earlier version of the method where "data NFRs" were not included in the metamodel, but organizational NFRs were (that do not affect the product itself, see Section 2.2.1). The corresponding metamodel that was the basis for these case studies can be found in [DKvK+03a]. In the SOL and ES case studies, the goal of the project was to transfer the complete NFR methodology to the respective company. The EMERGE case study was the only case study that used the automatic checklist generation, as the tool support was stable and mature enough to support this activity. Unfortunately, no case study could use the Elicitation Guide Tool, as they were performed before the Elicitation Guide Tool was stable and mature enough. With regard to the system domains, in the MBTech context the method was applied for the first time in an embedded context for a system that involves software and hardware parts. This case study was unique, as we were not sure whether quality models could be created as easily as for the pure software systems. It turned out that the concepts in the quality models worked well and the NFR methodology was also applicable in this kind of context. In contrast to this embedded domain, in the SOL case study, the method was tailored to be used with systems described mainly with business processes as the documentation entity representing the user tasks. In the other case studies, UCs were normally used for documenting the user tasks.

7.3 Validation Goals

This chapter refines the validation goals already stated in Section 1.2. The following hypothesis should be evaluated:

H1 - Feasibility: The elements of the NFR methodology are feasible, i.e., the artifacts can be created for real-life examples and the process activities can be applied by averagely trained personnel.

H2 -Completeness: The method results in a (more) complete set of NFRs. About 20% more critical NFRs are identified compared to the state of the practice. This is expected because the NFRs are elicited using experience-based quality models in a repeatable process.

H3 – Effort: The estimated rework effort in the subsequent project or platform development phases is reduced: The estimated saved rework effort for found NFRs is at least twice the effort spent on systematically eliciting the NFRs.

By using the GQM methodolgy [BCR94a], [BDR97], [vSB99] the goals are refined to the level of measures to evaluate the goals.

Goal 1: Feasibility

Analyze the NFR methodology

for the purpose of evaluation

with respect to feasibility

from the viewpoint of the method user

in the context of a series of 8 case studies.

Questions and Measures

Refining the goal into relevant questions that need to be answered enables the systematic definition of metrics to be calculated.

Q1.1: Is it possible to prioritize the set of HLQA?

Q1.2: Is it possible to build a quality model for more than one QA?

Q1.3: Is it possible to classify all QAs in the quality models according to the metamodel?

Q1.4: Is it possible to identify possible conflicts between QAs?

Q1.5: Is it possible to generate (manually or automatically) the checklists for the quality models?

Q1.6: Is it possible for the customers to answer the elicitation questions?

Q1.7: Is it possible to identify conflicts between the elicited NFRs?

Q1.8: Do the method users state that the method is usable?

Goal 2: Completeness

Analyze the set of NFRs resulting from the NFR methodology

for the purpose of evaluation

with respect to the desired completeness

from the viewpoint of the requirements engineer

in the context of a series of 8 case studies.

Questions and Measures

Q2.1: Can NFRs be identified by using the NFR methodology?

Measure M2.1: Number of NFRs elicited with the NFR methodology (entity evaluated: artifact)

Q2.2: Are more NFRs identified by using the NFR methodology?

Measure M2.2: Percentage of newly identified NFRs = M2.1*100% / Number of already existing NFRs (entity evaluated: artifact)

Q2.3: Do the elicited NFRs have a significant impact on the subsequent development phases (architecture)?

Measure M2.3: Expert criticality rating for each NFR for architecture relevance (entity evaluated: artifact)

Q2.4: Do the experience-based quality models help to make project-specific quality models more complete?

Measure M2.4: Were new QAs inserted into the project-specific quality models based on trigger questions from the experience-based QMs? (yes/no) (entity evaluated: artifact)

Q2.5: Were additional NFRs identified in the subsequent software development phases?

Measure M2.5: Number of additionally identified NFRs in the software development phases after the application of the NFR methodology (entity evaluated: artifact)

Goal 3: Effort

Analyze the NFR methodology

for the purpose of characterization and evaluation

with respect to effort

from the viewpoint of the method user

in the context of a series of 8 case studies.

Questions and Measures

Q3.1: How much effort is needed to perform the NFR methodology (total / per process activity)?

Measure M3.1: Effort in hours/days for the application of the NFR methodology (entity evaluated: process)

Measure M3.2: Effort in hours/days for the application of the NFR methodology spent by the domain expert, i.e., the person issuing the NFRs (activities P1.1, P1.2, P2.1) (entity evaluated: process)

Q3.2: How much rework effort is saved (in the architecture phase) by knowing the NFRs?

Measure M3.3: Estimated sum of effort in hours/days for rework if NFRs need to be incorporated into a product version without the NFRs (entity evaluated: process)

Q3.3: What is the ROI of the NFR elicitation?

Measure M3.4: M3.1/M3.3 (entity evaluated: process)

Question Q3.1 is used to characterize the NFR methodology; questions Q3.2 and Q3.3 are used to evaluate the NFR methodology.

7.4 Case Study Data and Results

Overall, the NFR methodology produced positive results. In the next sections, an analysis of the NFR methodology with regard to the feasibility of the method, the completeness of the set of NFRs, and the effort to perform the methodology is presented. This is followed by a summary of further qualitative or quantitative observations when performing the case studies.

7.4.1 Feasibility

The results of the feasibility analysis are summarized in Table 12.

Q1.1: Prioritizing HLQA

The first step of the methodology is the prioritization of the set of high-level quality attributes HLQA. Prioritization of quality attributes was performed in all case studies except the ES case study. In these seven case studies, it was possible to prioritize the set of HLQA by performing a stakeholder workshop using a common prioritization technique (six case studies) or by using a prioritization questionnaire (one case study). Therefore, step P1.1 of the methodology can be regarded as feasible.

Q1.2: Building Quality Models

One important goal of the NFR methodology was that it can be used for more than just one quality attribute. Therefore, an important aspect that needs to be demonstrated in order to show that this NFR methodology is feasible is that it is possible to build quality models for more than just one quality attribute. In all case studies except the Ricoh and GIS case studies, the NFR methodology was used for more than one quality attribute. In case of the Ricoh case study, the QA efficiency was the only relevant QA in scope for the NFR elicitation. In case of the GIS case study, security was the only QA in scope. With the NFR methodology, it was possible to build quality models for all QAs that were in scope for the NFR elicitation (see also Table 11 for a list of QAs in scope per case study), Figure 51 shows the distribution of the QAs. Therefore, building quality models for more than one QA in the NFR methodology can be regarded as feasible.

Q1.3: Classifying QAs

After quality models have been created, an essential next step is the classification of each quality attribute according to the metamodel (see

Project	Empress	Ricoh	GIS	FIN	MBTech	SOL	EMERGE	ES
1. Feasibility								
Q1.1: Is it possible to prioritize the set of HLQA?	yes	yes	yes	yes	yes	yes	yes	N/A
Q1.2: Is it possible to build a quality model for more than one QA?	yes	N/A	yes	yes	yes	yes	yes	yes
Q1.3: Is it possible to classify all QAs in the quality models according to the metamodel?	yes	yes	yes	yes	N/A	yes	yes	yes
Q1.4: Is it possible to identify possible conflicts between QAs?	N/A	N/A	N/A	N/A	N/A	yes	N/A	N/A
Q1.5: Is it possible to generate the checklists for the quality models?	yes	yes	yes, with manual adjustment	yes	yes, with manual adjustment	yes	yes, first automatic generation	N/A
Q1.6: Is it possible for the customers to answer the elicitation questions?	yes	yes	yes	yes	yes	N/A	yes	N/A
Q1.7: Is it possible to identify conflicts between the elicited NFRs?	yes, for a limited set	N/A	N/A	N/A	N/A	theoretically yes, but in practice no	N/A	N/A
Q1.8: Do the method users state that the method is usable?	yes	yes	yes	yes	yes	yes, except NFR conflicts	yes	N/A

Table 12: Summary of results for the feasibility goal

Section 2.2.2). Therefore, it is important to know whether it is possible to classify all quality attributes. In all case studies except the MBTech case study, the quality attributes were classified. In the MBTech case study, the system had only one system function and was treated as one system component. Therefore, a classification of the quality attributes was not needed, as all elicitation questions were asked for the same object. The case studies FIN, GIS, Ricoh, and Empress used an earlier version of the metamodel where "data NFRs" were not included. Therefore, there were difficulties in classifying quality attributes such as accuracy (of data). With the new metamodel, all quality attributes in the quality models can be classified. Therefore, the classification of quality attributes in the NFR methodology can be regarded as feasible.

Q1.4: Identifying QA Conflicts

In process activity P1.3, one checks for possible conflicts between QAs. There were three case studies, namely the EMRPESS, GIS, and SOL case studies, where the check for conflicting NFRs (P2.2) was in scope. In the SOL case study, NFRs were only elicited with examples, but not for a real project. The SOL case study was the only where process activity P1.3 was performed to prepare P2.2. In Empress and GIS, this effort was not in scope for the projects. For the SOL case study, quality attribute experts created complete dependency matrices. For the set of 59 quality attributes that were included in the quality models of the HLQA attributes efficiency, reliability, and security, 564 relationships were captured in these dependency matrices. It took a large amount of effort (about two person-days of effort per comparison between two HLQA attributes) to compare all quality attributes in the quality models. Therefore, one can say from this limited experience in one case study that the identification of QA conflicts is feasible, but that at least initially, high effort has to be invested.

Q1.5: Generating Checklists

In process activity P1.4, the checklists for the NFR elicitation are generated either manually or with the help of the Checklist Generation Tool. In all case studies except the ES case study, checklists were generated. In the EMERGE project, the Checklist Generation Tool was used for a project for the first time. The checklist derivation process was manual in the other six case studies. In the MBTech case study, the checklists were first created according to the algorithm and sentence patterns, but later on adjusted manually by the domain experts to better reflect domain-specific knowledge. For example, they added domain-specific example NFRs to the questions. In the GIS case study, too, examples were added to the checklist after the first creation according to the algorithm. In all other case studies, the checklist derivation process was straightforward and the checklists were easy to derive from the quality models. In some case studies, the metric information was added as additional information that makes the elicitation process easier and more accurate. As checklist generation was straightforward in most of the case studies, the checklist generation process can be regarded as feasible.

Q1.6: Answering the Questions

The actual NFR elicitation process activity P2.1 was performed for a concrete product in all case studies except the SOL and ES case studies. In these six case studies, the interviewed persons stated that it was possible to answer the questions asked by the checklists. In some of the case studies, it was necessary to explain the meaning of the quality attribute to the interviewed person. This was usually done by reading the definition of the QA from the quality model documentation. Therefore, the process of answering the questions can be regarded as feasible.

Q1.7: Identifying NFR Conflicts

As already stated in Q1.4, the process of NFR conflict identification was a subject in the Empress, GIS, and SOL case studies, but the structured process as depicted in process activity P2.2 was only used in the SOL case study. In Empress and GIS, the project could not follow this structured process in P2.2, as process activity P1.3 was not performed due to project effort limitations. In these two case studies, the process of NFR conflict identification was performed ad hoc on the set of elicited NFRs, which was quite small compared to the other case studies. In both case studies, conflicts were identified:

- In Empress, an intra-QA dependency conflict was detected, as two conflicting NFRs on the same QA (network throughput) were specified.
- In GIS, inter-QA dependencies were detected. When it came to making architectural decisions the elicited security NFRs were in competition with elicited usability and performance NFRs that were elicited later on.

In the SOL case study, process activity P2.2 was used, not for a concrete project, but for example material. As mentioned before, 564 relationships were captured in the dependency matrices. The calculations in Section 6.1.3 already show that a huge number of comparisons would have to be performed in order to systematically check all NFRs for conflicts. Therefore, the rating of the method users on the customer side was that for a large set of NFRs, the process could be applied, theoretically, but for effort reasons they do not think that this process activity is applicable in practice. Therefore, the identification of conflicts between NFRs can be regarded as feasible only for a small set of elicited NFRs or if a project has a small set of $QA \in QA_{InScope}$. For other cases, the identification of conflicts between NFRs is regarded as not feasible.

Q1.8: Overall Feasibility

In all case studies except the ES case study, which was interrupted due to project pressure, we asked the future method users whether they think the overall method is usable or not usable, or which limitations they see. All case studies except SOL and MBTech regarded the method as feasible as is. In the MBTech case study, the future method users suggested doing some workshop work as offline work (homework) first (see [ADB+08] for more details). Otherwise, they also confirmed the feasibility of the NFR methodology. In the SOL case study, the future method users stated that the method is feasible except for the identification of NFR conflicts.

The company that was the application partner in the SOL company dealt with this situation in the following way: The check of conflicts (P2.2) was taken out of the process for the company. A check of conflicts between NFRs was incorporated as an additional quality perspective into their perspective-based reading [BGL+96] inspection process. Some time after our technology transfer of the NFR methodology, our cooperation partners received a company award for the introduction of the NFR methodology in their company. To summarize, except for process activity P2.2, we can regard the NFR methdology as feasible.

Summary of Feasibility

To conclude this section, the last paragraph already summarizes the results of the feasibility evaluation very well: Except for P2.2, all parts of the NFR methodology are feasible from the viewpoint of the evaluated produced results as well as from the viewpoint of subjective opinions that were collected. The question of Q1.7 showed that P2.2 seems to be not feasible for larger sets of NFRs or large numbers of QAs. Therefore, future work is needed on this issue. Furthermore, one should take into account that as far as Q1.4 and Q1.7 are concerned, the evidence is very limited as they were only evaluated in few case studies.

7.4.2 Completeness

The results of the completeness analysis are summarized in Table 13.

Q2.1 Identification of NFRs

The first question to answer in order to analyze the completeness of the resulting set of NFRs is whether the application of the NFR methodology leads to an elicitation of new NFRs. Therefore, the absolute number of new NFRs was determined for all six case studies where NFRs were elicited. In four of them, the author had access to the resulting set of NFRs. In all of these four case studies, new relevant NFRs were identified by using the NFR methodology. The resulting sets of NFRs were always confidential to the project partners or to the consortium. Maybe at a later point in time, the deliverable of the EMERGE project containing the NFRs [GSJ+09] will be made publically available via the project website [EME09]. To give an impression of the distribution accross the various high-level quality attributes, the number of NFRs per quality attribute is provided where possible. In the MBTech and GIS case studies, we know that NFRs were identified, but not exactly how many. In the Empress case study, 56 new NFRs were identified. 12 of them were organizational NFRs, 5 were maintainability NFRs, 23 were efficiency NFRs, and 16 were reliability NFRs. In Ricoh, 16 new efficiency NFRs were elicited. In FIN, 44 new NFRs were identified, of which 21 were efficiency NFRs. Additionally, 7 reliability and 16 maintainability NFRs were elicited. In EMERGE, 104 new NFRs were elicited. 49 NFRs were UC or system function specific. Of these 49 NFRs, 21 were efficiency NFRs, 3 were reliablility NFRs and 25 were usability NFRs. 58 NFRs were requirements on the overall system, i.e., not specific for a system function or UC. Of these, 3 were efficiency NFRs, one was a reliability NFR, 45 were usability NFRs, 4

Project	Empress	Ricoh	GIS	FIN	MBTech	SOL	EMERGE	ES
2. Completeness								
Q2.1: Can NFRs be identified by using the NFR methodology?	yes, 56 new ones	yes, 16 new NFR	yes, number unknown	yes, 44 new NFR	yes, number unknown	N/A	yes, 104 NFR	N/A
Q2.2: Are more NFRs identified by using the NFR methodology?	56 new / 9 existing: +622%	16 new / 13 existing: +123%	N/A	44 new / 10 existing: +440%	yes, number unknown	N/A	N/A	N/A
Q2.3: Do the elicited NFRs have significant impact on the subsequent development phases (architecture)?	yes	N/A	yes	yes	N/A	N/A	yes	N/A
Q2.4: Do the experience-based quality models help in getting project specific quality models more complete?	yes	yes	yes	yes	yes	N/A	used exp. model as starting point	N/A
Q2.5: Were additional NFRs identified in the subsequent software development phases?	no	N/A	no	N/A	N/A	N/A	no	N/A

Table 13: Summary of results for the completeness goal

were security NFRs, and 2 were safety NFRs. So one can see that the NFR methodology is capable of identifying new NFRs for various quality attributes. In the conducted case studies, mostly efficiency and reliability NFRs were elicited.

Q2.2: Identification of Additional NFRs

A second step is to judge whether the application of the NFR methodology leads not only to the identification of NFRs, but also to additional NFRs that were not elicited before. Therefore, the percentage of newly identified NFRs is calculated. This was, of course, only possible for those case studies that had NFRs as part of the existing requirements specification, i.e., the Empress, Ricoh, and FIN case studies. In Empress, the requirements specification contained 9 NFRs prior to the systematic NFR elicitation. This corresponds to an increase of 622%. In the Ricoh case study, 13 NFRs existed before, which corresponds to an increase of 123%. In the FIN case study, 10 NFRs existed before, which corresponds to an increase of 440%. In the EMERGE case study, no NFRs were elicited before. One can conclude from these numbers that a significant number of additional NFRs were elicited by applying the NFR methodology.

Q2.3: Impact of NFRs

Having additional NFRs with low impact on the subsequent development phases would not be very beneficial. Therefore, an important question is whether these additionally elicited NFRs have a significant impact on the subsequent development phases such as the architecture phase. This was measured by asking for an expert criticality rating for each NFR regarding architecture relevance. In the Empress case study, the architects who developed the system prototype gave a rating for each elicited NFR on the importance for architecture. 23 of the elicited NFRs were rated as important for the current architecture; 9 of them were rated as critical for the current architecture, which means they would lead to an inevita-

ble change to the architecture. In the GIS case study, there was no systematic rating, but some of the elicited security NFRs had a strong impact on the architecture. In [DKK+05], it is stated: "An important observation that was made … was that the slightest changes to any of the NFR could have an important impact on the former architectural decisions (e.g., a decision was no longer a possible alternative as it no longer supports all of the requirements) thus requiring a new assessment of all possible alternatives." In the FIN case study, the elicited NFRs led to a major discussion between the product managers who issued the NFRs and the architects. In case these (especially efficiency) NFRs needed to be incorporated into the product, the complete architectural framework the product is based on would be obsolete and would need to be changed. Therefore, the trade-off analysis came to the conclusion that the NFRs will not be fulfilled in order to stay with the current architectural framework. In EMERGE, only a part of the elicited NFRs were incorporated into the research prototype as the incorporation of all NFRs would have required too much effort. The dimension of the effort to incorporate these NFRs for the EMERGE case study can also be seen in the data for Q3.2. All these case studies show that many of the elicited NFRs have a strong impact on the subsequent phases, leading either to architectural changes, new functionalities, or even completely new architectures.

Q2.4: Help of Experience-based Quality Models

A further question to answer is whether the experience-based quality models help to make the project-specific quality models more complete. Therefore, we determined whether new QAs were inserted into the project-specific quality models based on the trigger questions that came from the experience-based quality models. In other words, this also determines whether the project-specific quality models would have been incomplete if no experience-based quality models existed. In Empress and FIN, the experience-based quality models were used intensively to ask trigger questions. In Ricoh, the experience-based quality models were useful, but many domain-specific quality attributes were elicited that were not available in the experience-based quality models. In the MBTech case study, the experience-based quality models were useful, but the hardware-related quality attributes were elicited newly. In EMERGE, the experience-based quality models were not used to ask trigger questions, but directly as a starting point and discussion basis. Therefore, it is not possible to determine their usefulness compared to an approach where the project-specific quality models are created from scratch. To summarize, the case studies show that the experience-based quality models are an essential asset to make the project-specific quality models complete. On the other side, each project showed that process activity P1.2 revealed new domain-specific quality attributes that must be incorporated into the experience-based quality models. So process activity P1.2 should not be skipped.

Q2.5: Additional NFRs after NFR Elicitation

A final question that serves as an indicator for the analysis of completeness is whether NFRs were identified after the application of the NFR methodology. This would either mean that, due to a change in the domain experts' expectation, a new NFR became necessary, or that the NFR existed before, but was not identified during the NFR methodology. Therefore, we measure the number of NFRs that were identified after the application of the NFR methodology with M2.5. This data is only available for the Empress, GIS, and EMERGE case studies. In Empress and GIS, the software development for the product is finished. In EMERGE, a first running version of the research prototype is available. In all other case studies, we had no access to data on the subsequent software development processes, or the products are still in an early development phase. In all three case studies, no new NFRs emerged during the subsequent software development phases.

Summary of Completeness

To summarize this section, the case studies showed that the NFR methodology is capable of identifying NFRs. Furthermore, the NFR methodology is capable of identifying additional NFRs for projects where NFRs were already elicited with state-of-the-practice requirements engineering methods. The percentage of newly identified NFRs ranges from over 100% to over 600%. The case studies also showed that the elicited NFRs have a strong impact on the subsequent development phases, leading either to architectural changes, new functionalities, or even completely new architectures. This shows the relevance of the newly elicited NFRs. Additionally, the case studies showed that the experience-based quality models are an essential asset to make the project-specific quality models complete. One can assume that without experience-based quality models the project-specific quality models would not be complete. Of course, the case studies cannot show that the NFR methodology elicits a set of NFRs that is 100% complete. This proof is not possible, as for a real-life project, it is impossible to identify the 100% baseline. However, no new NFRs emerged during the subsequent software development phases in the three case studies where we had access to this information. Also, the NFR methodology shows that it elicits a far more complete set of NFRs than state-of-the-practice approaches. Whether it elicits a more complete set of NFRs than state-of-the-art approaches is probable but not yet proven. This analysis is subject to future work. As already stated in Section 7.1, this future work is not easy to perform, as one would either need to perform an experiment with a large real-life system or find an industrial partner that is willing to perform two state-of-the-art approaches for the same purpose.

7.4.3 Effort

The results of the effort analysis are summarized in Table 14.

Q3.1: Effort for Application

For judging the effort for the application of the NFR methodology, two measures were introduced. M3.1 measures the overall effort spent on the application. This is the effort that results from summing up

- the effort spent by the method experts on offline (preparation) activities and participation of the method expert in the process activities with the customer (P1.1, P1.2 and P2.1)
- the effort spent in the application by the domain expert (i.e., the person capable of issuing the NFRs), measured by M3.2.

In each case study, the data was recorded or estimated afterwards by the person applying the NFR methodology. The complete and detailed list of effort data for the case studies can be found in Appendix C.

In Table 15, more information on the spent effort is given. The overall effort required for the method application ranges from 6 person-days (Ricoh and EMERGE case studies) to 31 person-days (FIN case study). The average effort for applying the NFR methodology amounts to 14 person-days. When analyzing the effort spent by the domain experts, we can see that the case studies FIN (24 person-days) and MBTech (12 person-days) required most effort. This was due to the fact that many stakeholders from the customer organization attended the tailoring and elicitation workshops, respectively. Furthermore, in FIN the tailoring and elicitation activities P1.2 and P2.1 had a long duration. In EMERGE, the least effort was spent by the domain experts (1 day), as the tailoring and elicitation workshops were carried out in a very straightforward and concise manner. On average, the effort spent by all domain experts per case study amounts to 9 person-days.

Project	Empress	Ricoh	GIS	FIN	MBTech	SOL	EMERGE	ES
3. Effort								
Q3.1: How much effort is needed to perform the NFR methodology?	7 person-days	6 person-days	N/A	31 person-days	18 person-days	N/A	6 person-days	8 person-days
Q3.2: How much rework effort is saved (in the architecture phase) by knowing the NFRs?	15 person-days	N/A	N/A	N/A	N/A	N/A	103 person-days	N/A
Q3.3: What is the ROI of NFR elicitation?	ROI > 2	N/A	N/A	N/A	N/A	N/A	ROI > 17	N/A

Table 14: Summary of results for the effort goal

Project	Empress	Ricoh	GIS	FIN	MBTech	SOL	EMERGE	ES	Average*
Overall Effort (person-days)	7	6	N/A	31	18	N/A	6	8	14
Effort for Domain Experts (person-days)	5	3	N/A	24	12	N/A	1	5	9
Effort for Method Experts (person-days)	2	3	N/A	7	6	N/A	5	3	5
Effort for each Domain Expert (person-days)	1.5	1.5	N/A	3	3	N/A	0.2	1	1
Automation potential for Checklist Generator (person-days)	2	0.1	N/A	0.25	0.3	N/A	N/A	N/A	1
Automation potential for Elicitation Guide (person-days)	1	2	N/A	4	6	N/A	0.2	N/A	3
Sum of automation potential	3	2	N/A	4	6	N/A	0.2	N/A	3
Automation potential in %	43%	33%	N/A	13%	33%	N/A	3%	N/A	25%

*for the calculation of the average values, ES was not taken into account, as only P1.1 and P1.2 were performed, not P2.1

Table 15: Effort spent and automation potential

When looking at the effort that is required per domain expert, we see that the effort spent ranges from 0.2 person-days to 3 person-days, being 1 person-day on average. This should be a reasonable amount of effort to require from a domain expert for P1.2 and P2.1. These are encouraging data, as the domain experts are usually not available that much for the elicitation of requirements in general.

With regard to the effort of the method expert, we see that between 2 person-days (Empress) and 7 person-days (FIN) have been spent. This difference in effort is mainly due to the significantly different length of the activities P1.2 and P2.1. On average, 5 person-days of effort were spent by method experts in the case studies. The method experts' effort is the one that can be saved by using the tool support. One can see from Table 15 that the automation potential for the Elicitation Guide Tool is usually higher than the one for the Checklist Generation Tool. This is due to the fact that the time needed for creating the checklist is lower than that for the elicitation workshop. Overall, in the case studies between 3% and 43% of the overall effort spent, and 25% on average, can be saved by using the tool support.

Q3.2: Saved Rework Effort

In order to get an ROI estimation, we need to estimate the impact of the NFRs on the subsequent software development phases. For this purpose, the measure M3.3 is used. It is an expert estimate on the sum of effort in hours/days that is needed for rework if the newly elicited NFRs need to be incorporated into a product version that does not conform to the newly specified NFRs. There are two case studies where these values were provided by domain experts: Empress and EMERGE. The reason why it was possible to get this effort estimate in these case studies was the special design of the case studies: In EMPRESS, based on the requirements specification with the 9 existing NFRs, architects built a prototype of the system. In parallel, the NFR methodology was used to elicit the 56 new NFRs. After completion of the prototype, the architects were confronted with the newly elicited NFRs and asked for an estimate on

how much effort it would take them to fulfill the NFRs. They estimated this effort to three weeks of work (15 person-days). This was the effort estimate that holds for changing the prototype. We can assume that an effort estimate for a fully designed system would have been much higher. In EMERGE, NFRs were elicited, but due to time and resource constraints, there was a decision that only part of the NFRs would be incorporated into the research prototype. After creation of this research prototype, the domain expert responsible for architecture was asked to give an estimate on how much effort would be needed to completely incorporate the NFRs. The domain expert rated the overall effort needed to change the research prototype to reflect the remaining NFRs in the areas of Usability, Reliability, Security, and Safety at 102.5 person-days. The detailed effort estimates of the architect per quality attribute can be found in Appendix D. In all other case studies, the setup of the case studies unfortunately prevented a collection of this metric. Either no NFR elicitation was performed (SOL and ES), or no baseline system existed (GIS), or the method experts had no access to domain experts who could have provided effort estimates (Ricoh, FIN, MBTech). The data collected for Q2.3 shows that NFRs have a strong impact on the subsequent phases. This is another indicator that one can expect a significant reduction in rework effort for future case studies as well if NFRs are elicited systematically with the NFR methodology.

Q3.3 ROI of NFR Elicitation

From M3.3 and M3.1 we can calculate the Return on Investment from the application of the NFR methodology (M3.4). These values are only available for the Empress and EMERGE case studies. The calculation of M3.4 for Empress amounts to ROI = 15 days / 7 days = 2.14. We should keep in mind that this ROI is based on the value M3.3 for changing the prototype. Therefore, a higher ROI can be expected if the change had been estimated for a fully designed system. The calculation of M3.4 for EMERGE amounts to ROI = 103 days / 6 days = 17.17. The two case studies, especially EMERGE, show that the ROI value is promising. Nevertheless, one should keep in mind that the data is limited to the results of two case studies.

Summary of Effort

To summarize the evaluation of the NFR methodology with regard to the effort goal, we can say that sufficient data is available for characterizing the effort needed to perform the NFR methodology. The effort required for a single domain expert is one person-day on average. The effort for domain experts is 9 person-days on average, depending on the number of domain experts who are involved in the tailoring and elicitation workshops. This is a realistic effort to be expected from a customer. The effort for the overall application of the NFR methodology ranges from 6 to 31 days, being 14 person-days on average. Putting this into relation to typical efforts for requirements elicitation and specification in general, but also to the effort spent on functional requirements elicitation shows that these are realistic efforts to be spent. On average, 25% of the overall effort can be saved by using the existing tool support.

Limited, but positive experience exists with regard to the saved rework effort and ROI. Data was only collectable for two case studies. Therefore, the current result of ROI >2 and ROI > 17 is of low validity, but nevertheless encouraging. It seems that the efforts for eliciting the NFRs pay off, i.e., more effort can be saved in the subsequent software development phases.

7.4.4 Further Qualitative or Quantitative Observations

This section will describe additional interesting qualitative and quantitative findings that were observed during the case studies. A major quantitative observation here are the differences in measurability of the resulting NFRs in the case studies. A major qualitative observation are the conflicts that were found between NFRs in the case studies.

Measurability of NFRs

As already mentioned in 1.2, as a positive side-effect of the application of the NFR methodology, we expect the elicited NFRs to be measurable. In the case studies, we observed varying percentages of NFRs being measurable in the elicited sets of NFRs. The differences between the case studies and the reasons for these deviations will be demonstrated by the three case studies FIN, EMERGE, and Empress.

In the FIN case study, the statistics of NFRs were recorded as shown in Table 16.

	Overall NFRs	Measurable NFRs
Efficiency	21	21
Reliability	7	5
Maintainability	16	16
SUM	**44**	**42**

Table 16: Measurable NFRs in FIN

The percentage of measurable NFRs amounts to 95.5%. The two reliability NFRs that were stated as not being measurable refer to information that is collected at another place in the requirements specification (quantity structure of this software) and the value is subject to being calculated from this information. Once this information is present (this is to happen during the requirements engineering phase), the percentage of measurable NFRs amounts to 100%. One can see that this case study shows that the application of the NFR methodology can lead to a set of completely measurable NFRs.

In Empress, which is one of the early case studies, we see a slightly different analysis (see Table 17). 92.6% of the elicited NFRs are measura-

ble. An analysis of the reasons for the 7.4% non-measurable NFRs revealed that the moderators deviated from the elicitation process, i.e., the moderators did not ask the domain experts to state the NFRs by using the metric. Still, 92.6% is a very high value compared to the analysis of NFRs in other requirements specifications.

	Overall NFRs	Measurable NFRs
Organizational NFRs	12	10
Efficiency	23	22
Reliability	16	15
Maintainability	5	5
SUM	**56**	**52**

Table 17: Measurable NFRs in Empress

The last case study for which the analysis results are presented is the EMERGE case study. Table 18 shows the results for this case study.

In total, of the 104 stated NFRs, 72 were measurable, and 3 were partly measurable, i.e., one metric used for expressing the NFR was measurable, the other was not. This amounts to a percentage of 69.2 % measurable NFRs. An analysis of the reasons for the 30.8% not or partly measurable NFRs revealed the following reasons:

1. Not all elementary quality attributes had metrics attached: In the project-specific quality models that were used for the automatic checklist derivation, some elementary quality attributes had no metric attached. This is a clear deviation from the intended usage of the NFR methodology. As a result, the method expert asked the domain experts for NFRs and recorded their statements in a non-measurable way.

2. Method expert with domain knowledge: For the method expert, the sentence that phrased the NFR was measurable due to the domain knowledge that the method expert had acquired. But for the quality assurance person who checked the NFR for measurability, this domain knowledge was not present and therefore, the NFR was rated as not measurable. An example is the phrase "necessary information" that was specified as part of an NFR. The domain experts know that this is the information that is explicitly specified in the UC, but other persons might not know this.

3. Time pressure at the workshop: The elicitation workshops were performed with high time pressure. In this situation, it was not easy for

the moderator to stick to the process. Therefore, some NFRs were recorded without enforcing the use of a metric for this NFR.

	UC and System Function Specific NFRs (Overall / Measurable)	Non-UC and System Function Specific NFRs (Overall / Measurable / Partly Measurable)
Efficiency	21 / 21	3 / 3
Reliability	3 / 3	1 / 0
Usability	25 / 20	45 / 23 / 3
Security	-	4 / 2
Safety	-	2 / 0
SUM	**49 / 44**	**55 / 28 / 3**

Table 18: Measurable NFRs in EMERGE

These examples show that the percentage of measurable NFRs that will be achieved by the NFR methodology is dependent on two factors. First, the input for the parts that are tool-supported must be complete. If, for example, metrics are missing in the quality models that are input for the checklist derivation, this can have a strong impact on the resulting set of NFRs. Second, the parts that are moderated by a person can be influenced by the moderator's method as well as by domain knowledge and external factors such as time pressure. Therefore, these factors should also be systematically be taken into account and controlled when performing the NFR methodology with the goal of achieving measurable NFRs.

Conflicts between NFRs

As already mentioned in 1.2, as a positive side-effect of the application of the NFR methodology, we expect the set of elicited NFRs to be free of conflicts. In Section 7.4.1, some information was already presented for answering questions Q1.4 and Q1.7. For all eight case studies, we do not have any knowledge about whether conflicts between NFR have been detected in later software development phases. In SOL we experienced that a huge set of potential conflicts (>500) were elicited by systematically performing P1.3. (see Section 7.4.1, Q1.4). In the GIS and Empress case studies, we experienced the situation that elicited NFRs were detected as having a conflict in process activity P2.2. To give an example of such a conflict and the resulting solution, we refer to the Empress case study. There, an intra-QA dependency (see also Section 5.7) was detected. One NFR stated that the throughput for the wireless LAN should be limited to 11 MBit/sec. The rationale was to use inexpensive 11Mbit/sec standard hardware components. Another NFR stated for the throughput for the wireless LAN component that "In worst case 8 people shall be able to download 1 document per person within 5-10 secs.". When analyzing these two NFRs, the moderator asked for the typical size of such a document and found out that a typical document size of 3

MByte would amount to 8x3MByte=24MByte in 10 seconds. This means a necessary throughput of 2.4MByte/sec=19.2 MBit/sec, which is larger than the 11MBit/sec requested in the other NFR. As a solution, the maximum file size of the documents was restricted.

The fact that over 500 relationships showing possible conflicts were elicited in SOL and the necessary handling of conflicting NFRs in GIS and Empress show that conflicts between NFRs can appear quickly once the NFR elicitation is performed systematically and a more complete set of NFRs is specified.

7.5 Summary of Validation Results and Discussion

7.5.1 Summary of Results

The goal of the validation of the NFR methodology in eight case studies was to evaluate the method's feasibility, the completeness of the resulting set of NFRs, and the effort needed for the method's application. The validation setup would not allow determining statistically significant values for accepting or rejecting the corresponding null-hypothesis for the three evaluation goals. Rather than that, we give a qualitative statement on whether the data "support" or "strongly support" each hypothesis.

The results of the feasibility evaluation are very positive: Except for process activity P2.2, all parts of the NFR methodology are absolutely feasible from the viewpoint of the evaluated produced results as well as from the viewpoint of subjective opinions that were collected. Q1.7 showed that there seem to be some limitations regarding the feasibility of P2.2. This activity seems to be theoretically feasible, but not feasible in practice for larger sets of NFRs or large numbers of quality attributes due to the high amount of effort required. Furthermore, one should take into account that as far as Q1.4 and Q1.7 are concerned, the evidence is very limited, as these questions were only evaluated in few case studies.

In general, ***the data collected for Q1.1 to Q1.8 strongly support hypothesis H1 – Feasibility***: The elements of the NFR methodology are feasible, i.e., the artifacts can be created for real-life examples and the process activities can be applied by averagely trained personnel.

The result of the completeness evaluation is also very positive: The case studies showed that the NFR methodology is capable of identifying additional NFRs with a strong impact on the subsequent software development phases. This holds for projects where NFRs were already elicited with state-of-the-practice requirements engineering methods. The percentages of newly identified NFRs range from over 100% to over 600%. The case studies show that after the NFR methodology was applied, no

new NFRs emerged in the subsequent software development phases. One should take into account that the data collected for Q2.3 and Q2.5 are limited.

The data collected for Q2.1 to Q2.5 strongly support hypothesis H2 – Completeness: The method results in a (more) complete set of NFRs. The data definitively supports the statement that 20% more critical NFRs are identified compared to the state of the practice.

The result with regard to the effort needed to perform the NFR methodology is encouraging. The effort required for the domain experts is low (9 person-days on average), with an average of 1 person-day per domain expert. Also, the overall effort needed to perform the NFR methodology is low (14 days on average) and can be further decreased by using the existing tool support (25% on average). This effort is completely justifiable, especially if compared to the effort that can be saved in the subsequent phases by knowing the NFRs in advance: One case study showed that with the 7 days of effort invested, 15 days of rework effort could have been saved. This amounts to an ROI > 2. Another case study revealed that with 6 days of effort invested, 103 days of rework could have been saved, which amounts to an ROI > 17. One should take into account that sufficient data is available for evaluating the effort needed to perform the NFR methodology, but the experience with regard to the saved rework effort and ROI is very limited.

The data collected for Q3.1 to Q3.3 support hypothesis H3 – Effort: The estimated rework effort in the subsequent project or platform development phases is reduced: The estimated saved rework effort for found NFRs is at least twice the effort spent on systematically eliciting the NFRs.

As a positive side effect, we observed that the NFR methodology can produce a set of NFRs that is completely measurable. Important prerequisites to achieving this are the process discipline of the moderator and complete inputs in terms of quality models. As a further positive side effect, we experienced that the method can reveal conflicts in the set of elicited NFRs.

7.5.2 Threats to Validity

The NFR methodology was evaluated in eight case studies. Data was collected to evaluate the NFR methodology. Even though no controlled experiment was performed and no statistics were used to formally reject or accept the hypotheses, in this section, the possible threats to validity shall be discussed according to the four types of possible threats reported by [WRH+00].

Conclusion Validity

Based on the few data points collected in the eight case studies, a statistical analysis of the data was not appropriate. Furthermore, the data collected was collected "only" by performing eight case studies. We do not have any data from controlled experiments (see Section 7.1 for the reason). Therefore, a low statistical power is inherent.

Concerning the reliability of the measurements, most of the data collected is objective, i.e., two persons would interpret the results in the same way. Only for Q1.8 and Q3.2 were subjective measures used to get an expert judgement. For Q3.1, we have a special situation. Some effort data was recorded; other effort data was estimated by the person conducting the activity post mortem. The reliability of the estimations is lower than the recorded effort data. Large deviations from the estimated effort and the actual effort are unlikely. Slight deviations do not impact the conclusions we draw, as for the Empress example with the lower ROI value, the actual effort was recorded; for EMERGE, the ROI value is such high that a slight deviation would not impact our conclusion. The threats in these case studies are the estimates of the saved effort. For the other case studies, the effort data of Q3.1 is used not for ROI calculation but for characterizing the effort needed. Slight deviations in the post mortem estimations do not change the magnitudes that are of interest for the characterization.

Internal Validity

Threats to internal validity are influences that may affect the case studies with respect to causality and threaten the conclusion about possible causal relationships. We did not use statistics to prove causal relationships; rather the data collected support the hypotheses. Nevertheless, we want to discuss some possible threats to internal validity. The case studies might have been influenced by project circumstances that were not recorded. Such project circumstances might have been time pressure for the elicitation workshop as already reported for the EMERGE case study. We tried to observe this in each workshop. Other project circumstances that are much harder to observe could be reasons to not state NFRs due to group dynamics. We did not recognize such effects. Most likely, those NFRs would have been identified in the subsequent software development phases, which was not the case in the case studies.

Another possible threat is the new combination of experts in the elicitation workshops. Maybe NFRs would have come up without the NFR methodology, just by having exactly this group of experts together in one workshop. Due to the design as case studies, we cannot judge this effect. But since the same or similar experts recorded the NFRs in the state-of-the-practice approaches used in the projects before the application of the NFR methodology, we assume that this threat has a low probability.

A last threat to internal validity is the fact that the author of the NFR methodology performed some of the case studies himself (see Section 7.1 for the list of the case studies). This was part of the research method

to obtain feedback especially from the early case studies for the construction of the NFR methodology. Nevertheless, the author never applied the method alone, but always in a joint setting with another Fraunhofer employee. Later case studies were performed by persons other than the author in order to further eliminate this threat. One impact of the circumstance that the author was performing the methodology by himself might have been on the result of measurability of the set of elicited NFRs. If we compare the results of the case studies in Section 7.4.4, we see that the measurability of NFRs in EMERGE was obviously lower than in FIN or Empress. In EMERGE, the author was not part ot the method expert team. We know from asking the method expert in EMERGE that the reasons for lower measurability were less process discipline due to time pressure, incorrect process input (quality models without metrics), and domain knowledge of the method expert. It might be that the author has a tendency to stick more to the process, as it was developed by the author himself, and to compensate incorrect process input in the workshop. Even though the measurability of the resulting NFRs were only a positive side effect of the NFR method, it would be interesting to study these effects in future work to optimize the NFR methodology with regard to measurability.

Construct Validity

Threats to construct validity are conditions that limit the ability to draw conclusions from the case study results to the theory behind it. Here the question is whether the material used in the case studies is typical of the constructs used in the theory of the NFR methodology. The early case studies used material (quality models and checklists) that deviated from the final metamodel of the NFR methodology (see Section 7.2). The effects of this are very local: In the outcome, they only affect the organizational and data items and do not threaten the validity for all other types of functional conceptual elements. Further input material, such as the functional specification, originated from real-life systems, but conformed with the assumptions that the NFR methodology had for functional specifications. When comparing the outcome in theory with the observed outcome, we have the inherent problem that in theory we have the 100% complete set of NFRs, which we cannot determine as a baseline for the observed outcome. We used the indirect measure M2.5, the number of additionally identified NFRs in the software development phases after the application of the NFR methodology, in order to be able to determine missed NFRs. Although no NFRs emerged in the subsequent phases, we still do not know whether no NFRs existed in theory or whether they were missed throughout the software development. As this is rather improbable, we assume that the threat of judging completeness without knowing what is 100% is also quite low.

External Validity

Threats to external validity are conditions that limit the ability to generalize the results to practice. For the NFR methodology, the eight case case studies were performed in intentionally different settings: Two case studies were performed for small systems, two for medium systems, one for a medium to large system, and three case studies for large systems. Two

case studies were performed in the embedded domain, the other six were performed for interactive information systems. Different quality attributes were in scope for the method application. Overall, eight different quality attributes were treated in the eight case studies. Still, eight data points is very limited experience for generalizing the results into overall software development practice. In particular, the quality attributes robustness, portability, and safety were only treated in one case study. Also, only two case studies were conducted for the embedded domain.

7.5.3 Open Questions and Implications

The eight case studies were a good starting point for analyzing and supporting the hypotheses for feasibility, completeness, and effort. Still, some open questions remain, implying that future empirical work is needed.

With regard to ROI and influence on subsequent software development phases, first encouraging, but still very limited data was provided by the case studies. More empirical research is needed to obtain significant quantitative results. This can be achieved by performing more case studies where data on the subsequent software development phases is available. Another option would be to perform an experiment where two groups start developing from different sets of NFRs: One group gets almost no NFRs, another group gets a full-fledged set of NFRs. After designing a solution for these requirements, the group with almost no NFRs will receive a change request to incorporate the missing NFRs and the effort for this change will be measured. Such an experiment could reveal solid data, but unfortunately, it is quite effort-consuming to built such a system, as the size of such systems must be reasonably large (see also Section 7.1).

The case studies pointed out that checking for NFR conflicts (process activity P2.2) is theoretically possible, but might be a task that requires lots of effort in practice. NFR conflicts were only identified for a medium-sized system (Empress). For the large system in SOL, the rating came from an expert judgment based on example material. Therefore, a case study that uses a large, real-life example with elicited NFRs would be a good starting point for gathering quantitative data on how many comparisons are needed for a real-life, large project.

The data from the eight case studies support the statement that the application of the NFR methodology results in a much more complete set of NFRs than state-of-the-practice approaches. Comparisons to other state-of-the-art approaches still need to be evaluated. One comparison of this NFR metholdolgy to the MOQARE approach can be found in a paper [HKD07]. However, more empirical research would be needed to allow a comparison with other state-of-the-art approaches. Designing a

controlled experiment to compare two or more approaches would again encounter the same obstacles as already described in Section 7.1. Therefore, one could design case studies where, for the same system, domain experts perform first one state-of-the-art approach for eliciting NFRs, and then the NFR methodology. This should also be performed the other way round. One would evaluate whether one method finds NFRs that were not elicited with the other method. Unfortunately, it is hard to find domain experts who are willing to perform process activities with the same purpose (eliciting NFRs for their system) twice.

The eight case studies definitively showed that the NFR methodology is applicable for more than one quality attribute. Some quality attributes such as efficiency and reliability were often addressed in the case studies (see Section 7.1). Others like safety, portability, or robustness were only treated once. The applicability of the NFR methodology for these quality attributes and maybe others that were not the subject of any of the eight case studies would be an issue for future case studies.

Another open question for future empirical research is where the differences in measurability of the elicited NFRs originate from and how to eliminate these differences. Here controlled experiments could be performed. One could vary the quality of the input artifacts or train the moderator in different ways to achieve differences in process discipline. Even though a deeper understanding of the effect of process discipline on the measurability of NFRs would be interesting, the problem can partially be solved with the existing tool support, as the amount of human activity is already reduced by automating process activity P2.1 with the Elicitation Guide Tool.

8 Summary

In this chapter, we will summarize and discuss the major results and contributions of this thesis and provide some ideas for potential future work based on existing method limitations.

8.1 Results and Contribution

Completeness of NFRs

Requirements engineering is the first activity in engineering a software-based product. Making mistakes in such an early phase has a strong impact on the subsequent software development phases. Besides the functional requirements, non-functional requirements play an important role for the success of a project or product. In today's practice, essential information on the system's non-functional requirements has often not been elicited properly and is thus incomplete. As a result, architectures have to change in late development phases, which leads to increased project or platform development costs and increased time to market. Alternatively, missing NFRs are not incorporated into the product in later phases, leading to low product quality.

Scope

This thesis addresses the topic of complete non-functional requirements elicitation. It focuses on NFR elicitation and specification for software-based, interactive systems. The scope is to provide support for as many quality attributes as possible, i.e., without restriction to one specific quality attribute (such as efficiency or usability).

Current Approaches

Current state-of-the-practice approaches are mainly based on free brainstorming on chapters of requirements specifications. Also, current state-of-the-art approaches treat NFRs in parallel to functional specifications. Neither the state-of-the-practice nor the state-of-the-art approaches offer a possibility to judge the completeness of the NFR elicitation. This is due to the fact that no objective end criterion is defined for the NFR elicitation process.

Idea of this Thesis

The key idea of this thesis is the systematic elicitation of NFRs taking specific elements of the functional specification as input and algorithmically processing the functional specification elements. By systematically processing the elements of the functional specification, the process becomes repeatable and controllable, which is the main driver for increasing the confidence that all important NFRs have been identified. Furthermore, the experience-based quality models provide a classified hierarchy of quality aspects. In the systematic NFR elicitation, the functional

conceptual elements and subsystems are checked against these quality aspects.

Contribution of the NFR Methodology

The NFR methodology described in this thesis provides a systematic approach for the elicitation, analysis, and specification of a complete set of NFRs. As a positive side effect, the set of NFRs is conflict-free and each NFR is measurable. To achieve this, the NFR methodology contributes the following components:

- A requirements taxonomy and metamodel (see Sections 2.2.1 and 2.2.2) incorporating functional, non-functional, and architectural concepts and the relationships between these concepts: The metamodel lays the foundation for the specification of NFRs as well as for the elicitation algorithm.
- A representation for quality models (see Section 2.2.3): This representation captures hierarchical, classified quality attributes and their dependencies.
- A guideline for the integrated specification of functional and non-functional requirements (see Section 3.2.3): A recommendation is given as to where to specify the different kinds of NFRs that can emerge during the elicitation process.
- An algorithm for eliciting NFRs based on functional specification elements and quality attributes (see Section 4.2): The elicitation algorithm takes into account the various relationships between elementary quality attributes and functional conceptual elements and subsystems, respectively. It provides the basis for the effective and efficient manual and tool-supported NFR elicitation.
- Complete and detailed process guidance (see Sections 5.1-5.8): Detailed process guidance is given on how to use the elicitation algorithm, all involved artifacts, and the tool support for NFR elicitation and specification.
- Checklists and tool support (see Sections 4.3 and 6.2): The checklists provide support for the manual application of the elicitation algorithm. The tool support partially automates the NFR methodology. The Checklist Generation Tool generates checklists automatically from the quality models with the help of sentence patterns. The method expert's part of elicitation activity P2.1 can be automated with the help of the Elicitation Guide Tool.
- Means to check the resulting set of NFRs for conflicts (see Sections 2.2.3.3 and 5.7): The NFR methodology makes use of influence relationships between QAs to later on check the resulting set of NFRs for conflicts. Intra-QA and inter-QA dependency checks are performed to identify conflicting NFRs.
- Means to focus the effort for NFR elicitation (see Section 6.1): Especially for large projects, an elicitation of a complete set of NFRs might not be desirable due to effort restrictions. In such situations, it is essential to ensure that the elicitation of NFRs does not reveal arbitrary

NFRs but still focuses on a systematic process. Therefore, the NFR methodology provides means for focusing on the critical qualities, functionalities, and subsystems. This also includes the possibility to perform the NFR methodology iteratively (see Section 6.1.4).

Evaluation of the NFR Methodology

To validate some of the potential benefits, the NFR methodology was deployed in a series of eight, mainly industrial case studies. The case studies varied in many dimensions, including the domain of the product, the size of the system, and the QAs in scope for NFR elicitation. The validation was driven by the hypotheses stated in Section 1.2. The results of the case studies strongly support the first two hypotheses:

- H1 – Feasibility: "The elements of the NFR methodology are feasible, i.e., the artifacts can be created for real-life examples and the process activities can be applied by averagely trained personnel." The analysis revealed that all elements of the NFR methodology are theoretically feasible in the various case study contexts. The practical feasibility was also given for all process activities, except for process activity P2.2. There, domain experts judged that for large systems, this activity might consume an impractical amount of effort.
- H2 – Completeness: "The method results in a (more) complete set of NFRs. About 20% more critical NFRs are identified compared to the state of the practice." In all case studies that performed activity P2.1, new, important NFR were elicited that were not elicited with the state-of-the-practice approaches. The ratio of newly identified NFR range from over 100% to 622%.

Furthermore, the results of the case studies are encouraging with regard to the last hypothesis:

- H3 – Effort: "The estimated rework effort in the subsequent project or platform development phases is reduced: The estimated saved rework effort for found NFRs is at least twice the effort spent on systematically eliciting the NFRs." The effort data shows that the effort for NFR elicitation is reasonably low. Only limited, but encouraging data (two case studies) exist for the ROI evaluation due to case study restrictions. In these case studies, the NFR methodology application resulted in an ROI > 2 and ROI > 17, respectively.

As a positive side effect, the case studies showed that the resulting set of specified NFRs can be measurable. Rates of 95.5% and even up to 100% can be achieved by consequently using the NFR methodology. Furthermore, the concepts of the NFR methodology revealed their potential to detect conflicts between NFRs in the resulting set of NFRs.

Tool Support

Some parts of the NFR methodology are intentionally constructed in an algorithmic or pattern-based way so that they can be easily automated. The NFR methodology is supported by two tools: the Checklist Genera-

tion Tool and the Elicitation Guide Tool. The Checklist Generation Tool makes manual derivation of checklists obsolete. The Elicitation Guide Tool automatically guides the domain expert by following the elicitation algorithm and asking for NFRs, while showing the corresponding functional conceptual elements and subsystems, respectively. This leads to saved effort, as the part of the method expert is taken over by the tool, but also to less sources for mistakes in performing the NFR methodology and to high repeatability.

The case studies showed that the automation potential from using the tool is high. On average, 25% of the overall effort could have been saved by using the tool support. The tool support also enables and supports the iterative usage of the NFR methodology. The scope of the NFR methodology with regard to qualities or functionalities can first be reduced and later on extended without unnecessarily repeating any previous elicitation activities.

8.2 Method Limitations and Future Work

Method Limitations and Future Work

In the case studies, the NFR methodology was used for the following QAs: usability, reliability, maintainability, portability, efficiency; security, robustness, and safety. Some QAs were treated more often than others. For some QAs, it was easier to create the quality models than for others. We faced most difficulties with safety. It is very hard to define suitable elementary quality attributes and especially metrics for safety. This can also be seen in the work by [BKL+95], where safety is treated differently than the other QAs. Furthermore, safety is the only QA among the others that ISO 9126 [ISO01] puts on the "quality in use" level. The working group "Non-Functional Requirements" of the German Computer Society (GI) [Doe09] has created quality models for many QAs, but did not succeed in building a model for safety that could be agreed upon. Therefore, for safety-critical systems, we recommend addressing safety with other methods to ensure complete coverage of this important topic.

For those QAs where we know that quality models can be produced, an interesting question to pursue in the future is on which basis reference quality models can be created for companies. Maybe some domain-specific patterns exist that would allow fast creation of the reference quality models. More research is needed to analyze whether this kind of patterns exist.

As the NFR methodology incorporates generic mechanisms for treating different kinds of tasks, data items, and subsystems, it can be applied to small systems as well as to large-scale systems. In our case studies, we experienced that the QAs that characterize items on an organizational business-process level may be different from QAs that affect items on the IT-system level (like, for example, the typical UC level). Therefore, fu-

ture work should investigate whether there are quality attributes in reference quality models that are more suitable for the business-process level and others that are more suited for the IT-system level. Maybe an additional classification of quality attributes could enable additional focusing in the elicitation process.

Experience has shown that checking for inter-QA dependencies is a non-trivial task (see Sections 6.1.3, 7.4.1 and 7.5.3). This is due to the large number of comparisons that is needed to completely check all possible NFRs that might be in conflict, even though the number of comparisons is reduced by focusing on those NFRs where the QAs, the NFRs are type of, stay in conflict. Additional strategies are needed to make this activity more feasible.

This NFR methodology was designed to systematically elicit NFRs. Parts of the approach can be automated. Currently, automation is only possible for the activities of the method expert. Domain experts are still needed for the tailoring and the elicitation. Even though the effort is already quite low, future work should aim at further reducing the effort required by the domain experts, as they are usually the bottleneck for eliciting the requirements. This could be done, for example, by inviting specific domain expert roles to the tailoring workshops depending on the part of the quality model that is subject to tailoring instead of doing the tailoring in a workshop with all domain experts. But then, one has to ensure that this does not negatively influence the completeness of the project-specific quality models.

The NFR methodology today foresees reuse on the level of QAs. We know from our industrial case studies that some domain experts would appreciate reuse also on the set of concrete NFRs stated for previously built, similar products. Other work also incorporates such reuse approaches [CL01a], but approaches purely building upon NFR reuse are typically not scalable. Therefore, future work could investigate the effect of incorporating reusable examples of NFRs into the NFR methodology and especially into the Elicitation Guide Tool. This could also reduce the effort required by the domain experts.

Limitations in Empiricism and Future Work

The NFR methodology was evaluated in a series of eight case studies. This analysis was a good starting point for analyzing and supporting the hypotheses for feasibility, completeness, and effort. Still, some open questions remain, implying that future empirical work is needed. More empirical research is needed to get significant quantitative results for the method's ROI and the impact of NFRs on the subsequent software development phases.

Furthermore, with regard to the feasibility of the NFR dependency check, a case study that uses a large, real-life example with elicited NFRs would be a good starting point for gathering quantitative data on how many

comparisons are needed for a real-life, large project to detect NFR conflicts.

To further evaluate the performance of the NFR methodology, more case studies comparing the NFR methodology to other state-of-the-art approaches should be conducted. The eight case studies definitively showed that the NFR methodology is applicable for more than one quality attribute. Some quality attributes such as efficiency and reliability were often addressed in the case studies (see Section 7.1). Others like safety, portability, or robustness were only treated once. The applicability of the NFR methodology for these quality attributes and maybe others that were not the subject of any of the eight case studies could be the subject of evaluation in future case studies.

Last but not least, there was high variance in the results on the measurability of the set of NFRs. Here, controlled experiments could be performed to get a deeper understanding of the effect of process discipline on the measurability of NFRs.

Tool Limitations and Future Work

This thesis uses primary and secondary information places for annotating NFRs. This is due to the fact that information needs are different for the various roles and, therefore, there is not one ideal solution: The need differs from NFRs to be visualized in every view to visualization solely at the subsystem or interaction descriptions. Future tool support should have a feature to enable and disable the NFR information in the various views. This would definitely be a great help in addressing this challenge of view-based NFR documentation.

Furthermore, for checking NFR conflicts, tool support would be beneficial. This tool support could bring the candidate pairs of possibly conflicting NFRs to the attention of the requirements analyst and ask whether the two NFRs really stay in conflict. This would take the burden of searching for the candidate pairs off the method and domain experts.

The current Elicitation Guide Tool is based on IBM Telelogic Doors. Performing business process modeling with Doors when modeling large-scale information systems is possible with the Analyst Plug-in, but it is not efficient. Typically, a business process or workflow-modeling tool would be used. Therefore, it would be beneficial to port the existing functionality from the Elicitation Guide Tool also to a workflow-modeling tool.

8.3 Concluding Remarks

The research conducted as part of this thesis is a classical example of applied research: The motivation for the research originated from industrial practice. In our industry projects, we encountered the situation that NFRs

were insufficiently elicited and specified. We surveyed the state-of-the-art approaches with regard to NFR elicitation, specification, and modeling and found no approach that enables systematic elicitation of NFRs to obtain a complete set of NFRs. Most approaches were not feasible for industrial application, or few statements about the effort required to perform the approaches existed. Therefore, based on the existing state-of-the-art approaches, we developed new ideas, including the key idea of algorithmically processing certain types of functional elements. We designed all the aforementioned components of the NFR methodology including the tool support. Eventually, we validated the researched methodology in eight, mainly industrial settings to provide insights into the method's feasibility, NFR completeness, and effort needed.

We know that dealing with NFRs is an important topic and the research in this thesis contributed significantly to the challenge of getting complete NFRs. We still see many research challenges to be addressed in the future to enable our industry to deliver high-quality systems within a reasonable amount of effort and time.

References

[AD07a] S. Adam and J. Doerr, *On the Notion of Determining System Adequacy by Analyzing the Traceability of Quality*, 8th Workshop on Business Process Modeling, Development, and Support (BPMDS'07) in conjunction with Caise, 2007.

[AD07b] S. Adam and J. Doerr, *Towards Early Consideration of Non-Functional Requirements at the Business Process Level*, IRMA Conference, 2007.

[ADB+08] S. Adam, J. Doerr, F. Blucha and A. Poth, *High Quality in Elicitation and Specification of Non-functional Requirements - Lessons Learned from Applying this Method to the Automotive Domain*, CONQUEST - 11th International Conference on Quality Engineering in Software Technology, 2008, pp. 123-132.

[ADE+09] S. Adam, J. Doerr, M. Eisenbarth and A. Gross, *Using Task-oriented Requirements Engineering in Different Domains - Experiences with Application in Research and Industry*, 17th IEEE International Requirements Engineering Conference, 2009.

[Ant97] A. Antón, *Goal Identification and Refinement in the Specification of Information Systems*, PhD Thesis, Georgia Institute of Technology, 1997.

[ARD09] S. Adam, N. Riegel and J. Doerr, *The Role of Quality Aspects for the Adequacy of Business Processes and Business Information Systems*, International Journal of Business Process Integration and Management, vol. 4, no. 2, 2009, pp. 124-133.

[Bas93] V.R. Basili, *The Experimental Paradigm in Software Engineering*, Proceedings of the International Workshop on Experimental Software Engineering Issues: Critical Assessment and Future Directions, 1993.

[BB02] J.E. Burge and D.C. Brown, *NFR's: Fact or Fiction?*, Computer Science Technical Report, WPI-CS-TR-02-01, Worcester Polytechnic Institute, 2002.

[BBF+01] P. Botella, X. Burgues, X. Franch, M. Huerta and G. Salazar, *Modeling Non-Functional Requirements*, Jornadas de Ingenieria de Requisitos Aplicada (JIRA), 2001.

[BBK+78] B.W. Boehm, J.R. Brown, H. Kaspar, M. Lipow, G.J. Macleod and M.J. Merrit, *Characteristics of Software Quality*, North-Holland, 1978.

[BBL76] B.W. Boehm, J.R. Brown and M. Lipow, *Quantitative evaluation of software quality*, International Conference on Software Engineering, 1976.

[BCR94a] V. Basili, G. Caldiera and H.D. Rombach, *The Goal Question Metric Approach*, Encyclopedia of Software Engineering, John Wiley & Sons, 1994, pp. 528–532.

[BCR94b] V.R. Basili, C. Caldiera and H.D. Rombach, *Experience Factory*, Encyclopedia of Software Engineering, 1, J. J. Marciniak, ed., John Wiley & Sons, 1994, pp. 469-476.

[BDA04] V. Basili, P. Donzelli and S. Asgari, *A Unified Model of Dependability: Capturing Dependability in Context*, IEEE Software, vol. 21, no. 6, 2004, pp. 19-25.

[BDK+99] J. Boegh, S. Depanfilis, B. Kitchenham and A. Pasquini, *A Method for Software Quality Planning, Control, and Evaluation*, IEEE SW, vol. 23, no. 2, 1999, pp. 69-77.

[BDR97] L.C. Briand, C. Differding and H.D. Rombach, *Practical guidelines for measurement-based process improvement*, Software Process: Improvement and Practice Journal, vol. 2, no. 4, 1997.

[Beu00] L. Beus-Dukic, *Non-functional requirements for COTS software components*, International Conference on Software Engineering, 2000.

[BGL+96] V.R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull and S. Sørumgård, *The Empirical Investigation of Perspective-Based Reading*, Empirical Software Engineering Journal, vol. 1, no. 2, 1996, pp. 133-164.

[BGR09] R.B. Svensson, T. Gorschek and B. Regnell, *Quality Requirements in Practice: An Interview Study in Requirements Engineering for Embedded Systems*, Requirements Engineering: Foundation for Software Quality (REFSQ): 15th International Working Conference, 2009, pp. 218-232.

[BH96] B. Boehm and H. In, *Identifying Quality Requirement Conflicts*, IEEE Software, vol. 13, no. 2, 1996.

[BKL+95] M. Barbacci, M.H. Klein, T.A. Longstaff and C.B. Weinstock, *Quality Attributes*, CMU/SEI-95-TR-021, 1995.

[BL91] V. Berzins and Luqi, *Software Engineering with Abstractions*, Addison-Wesley, 1991.

[BLF02] K.K. Breitman, J.C.S.d.P. Leite and A. Finkelstein, *The world's a stage: a survey on requirements engineering using a real-life case study*, Pontificia Universidade Catolica do Rio de Janeiro Departamento de Informatica, 2002.

[BMA02] I. Brito, A. Moreira and J. Araujo, *A Requirements Model for Quality Attributes*, Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, Workshop at 1st International Conference on Aspect-Oriented Software Development, 2002.

[Boe08] J. Boegh, *A New Standard for Quality Requirements*, IEEE Software, vol. 25, no. 2, 2008, pp. 57 - 63.

[Boe81] B.W. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.

[BR88] V.R. Basili and H.D. Rombach, *The TAME project: towards improvement-oriented software environments*, IEEE Transactions on Software Engineering, vol. 14, no. 6, 1988, pp. 758-773.

[Bro87] F.P. Brooks, *No silver bullet: essence and accidents of Software Engineering*, IEEE Computer, vol. 20, no. 4, 1987, pp. 10-19.

[BSD+07] T.H.A. Balushi, P.R.F. Sampaio, D. Dabhi and P. Loucopoulos, *ElicitO: A Quality Ontology-Guided NFR Elicitation Tool*, International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ), 2007.

[Bus09] F. Buschmann, *Learning from Failure, Part 1: Scoping and Requirements Woes*, IEEE Software, vol. 26, no. 6, 2009 pp. 68 - 69.

[CKM01] J. Castro, M. Kolp and J. Mylopoulos, *Towards requirements-driven information systems engineering: the Tropos project*, Information Systems Journal, vol. 27, no. 6, 2001.

[CL01a] L.M. Cysneiros and J.C.S.d.P. Leite, *Driving Non-Functional Requirements to Use Cases and Scenarios*, XV Brazilian Symposium on Software Engineering, 2001.

[CL01b] L.M. Cysneiros and J.C.S.d.P. Leite, *Using the Language Extended Lexicon to Support Non-Functional Requirements Elicitation*, Workshop em Engenharia de Requisitos, 2001, pp. 139-153.

[CL01c] L.M. Cysneiros and J.C.S.d.P. Leite, *Using UML to reflect non-functional requirements*, Conference of the Centre for Advanced Studies on Collaborative Research, 2001.

[CL99] L.M. Cysneiros and J.C.S.P. Leite, *Integrating Non-Functional Requirements into data modeling*, Proc. 4th International Symposium on Requirements Engineering, 1999.

[CM78] J.P. Cavano and J.A. McCall, *A Framework for the Measurement of Software Quality*, Software Quality and Assurance Workshop, ACM Special Interest Group on Measurement and Evaluation, 1978.

[CMB08] J. Cleland-Huang, W. Marrero and B. Berenbach, *Goal-Centric Traceability: Using Virtual Plumblines to Maintain Critical Systemic Qualities*, IEEE Transactions on Software Engineering, vol. 34, no. 5, 2008, pp. 685-699.

[CNY95a] L. Chung, B. Nixon and E. Yu, *Using Non-Functional Requirements to Systematically Select Among Alternatives in Architectural Design*, 1st International Workshop on Architectures for Software Systems, 1995, pp. 31-43.

[CNY+99] L. Chung, B.A. Nixon, E. Yu and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, 1999.

[Coc00] A. Cockburn, *Writing Effective Use Cases*, Pearson, 2000.

[CY98] L. Chung and E. Yu, *Achieving System-Wide Architectural Qualities*, OMG-DARPA MCC Workshop on Compositional Software Architectures, 1998.

[Dav93] A. Davis, *Software Requirements: Objects, Functions and States*, Prentice Hall, 1993.

[DFvL91] R. Darimont, S. Fickas and A.v. Lamsweerde, *Goal-Directed Concept Acquisition in Requirements Elicitation*, IWSSD-6, 6th International Workshop on Software Specification and Design, 1991, pp. 14-21.

[DHK+07] J. Doerr, S. Hartkopf, D. Kerkow, D. Landmann and P. Amthor, *Built-in User Satisfaction - Feature Appraisal and Prioritization with AMUSE*, 15th IEEE International Requirements Engineering Conference, 2007, pp. 101-110.

[DKK+03] J. Doerr, D. Kerkow, T. Koenig and T. Olsson, *A Method for Eliciting, Documenting, and Analyzing Non-functional Requirements*, Reportnr. 141.03/E, Fraunhofer IESE, 2003.

[DKK+05] J. Doerr, D. Kerkow, T. Koenig, T. Olsson and T. Suzuki, *Non-Functional Requirements in Industry - Three Case Studies Adopting an Experience-based NFR Method*, 13th IEEE International Requirements Engineering Conference, 2005.

[DKK+06] J. Dörr, D. Kerkow, T. Koenig and T. Olsson, *Qualität in Software & Systemen - Ein praxiserprobter Ansatz zur Erhebung und Spezifikation von Nichtfunktionalen Anforderungen – und was kommt jetzt?*, Softwaretechnik-Trends, vol. 26, no. 1, 2006.

[DKL+08] J. Doerr, D. Kerkow, D. Landmann, C. Graf, C. Denger and A. Hoffmann, *Supporting requirements engineering for medical products: early consideration of user-perceived quality*, 30th International Conference on Software Engineering, 2008.

[DKvK+03a] J. Dörr, D. Kerkow, A. von Knethen and B. Paech, *Eliciting Efficiency Requirements with Use Cases*, 9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ), Workshop held at CaiSE, 2003.

[DKvK+03b] J. Dörr, D. Kerkow, A. von Knethen and B. Paech, *Auf dem Weg zu messbaren Wartungsanforderungen*, Softwaretechnik-Trends, vol. 23 no. 1, 2003.

[Doe09] J. Doerr, *Homepage of the GI working Group on NFRs*, 2009, http://www.re-wissen.de/Arbeitsgruppen/AK-NFR/, last accessed 2009-12-28.

[DOS04] J. Doerr, T. Olsson and K. Schmid, *Qualität im Automobil: Systematische Definition nichtfunktionaler Anforderungen*, Proc. Automotive - Safety & Security, 2004.

[Ebe98] C. Ebert, *Putting requirement management into praxis: dealing with non-functional requirements*, Information and Software Technology, vol. 40, no. 3, 1998, pp. 175-185.

[EME09] *Webpage of the EMERGE project*, http://www.emerge-project.eu/, last accessed 2009-12-10.

[Eva95] *Evaluation of natural language processing systems*, 1995, http://www.issco.unige.ch/ewg95, last accessed 2009-12-29.

[EYM06] N.A. Ernst, Y. Yu and J. Mylopulous, *Visualizing non-functional requirements*, Proc. First International Workshop on Requirements Engineering Visualization, IEEE, 2006.

[FC03] X. Franch and J.P. Carvallo, *Using quality models in software package selection*, IEEE Software, vol. 20, no. 1, 2003, pp. 34-41.

[FD96] A. Finkelstein and J. Dowell, *A Comedy of Errors: The London Ambulance Service Case Study*, International Workshop on Software Specifications and Design, 1996, pp. 2-5.

[Fra98] X. Franch, *Systematic Formulation of Non-Functional Characteristics of Software*, International Conference on Requirements Engineering, 1998.

[Gar88] D.A. Garvin, *Managing Quality: The Strategic and Competitive Edge*, NY: The Free Press, 1988.

[Gil05] T. Gilb, *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*, Butterworth-Heinemann, 2005.

[Gil07] K. Gilb, *Evolutionary Project Management and Product Development, Book Manuskript to be obtained at* www.gilb.com, 2007.

[Gli05] M. Glinz, *Rethinking the Notion of Non-Functional Requirements.*, Third World Congress for Software Quality (3WCSQ 2005), 2005, pp. 55-64.

[Gli07] M. Glinz, *On Non-Functional Requirements*, 15th International Conference on Requirements Engineering, 2007.

[Gra92] R.B. Grady, *Practical Software Metrics for Project Management and Process Control*, Prentice Hall, 1992.

[GRL02a] *GRL Tutorial*, University of Toronto, 2002.

[GRL02b] *GRL Syntax*, University of Toronto, 2002.

[GS05] A. Gregoriades and A. Sutcliffe, *Scenario-based assessment of nonfunctional requirements*, IEEE Transactions on Software Engineering, vol. 31, no. 5, 2005, pp. 392- 409.

[GSJ+09] A. Gross, S. Steinbach-Nordmann, A. Jedlitschka, M. Becker, I. Steinke and M. Bloice, *D6.1.2 (System Requirements Specification), v2.0, EMERGE project, internal report*, 2009.

[Hae05] P. Haefele, *Softwareentwicklung mit dem TRAIN-Prozess*, Bachelor thesis, Institut für Informatik, Lehrstuhl Software Systeme, Universität Heidelberg, Heidelberg, 2005.

[HKD07] A. Herrmann, D. Kerkow and J. Doerr, *Exploring the Characteristics of NFR Methods – a Dialogue about two Approaches*, International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ), 2007.

[HP08] A. Herrmann and B. Paech, *MOQARE: misuse-oriented quality requirements engineering*, Requirements Engineering Journal, vol. 13, no. 1, 2008, pp. 73-86.

[IBR+01] H. In, B. Boehm, T. Rodgers and M. Deutsch, *Applying WinWin to quality requirements: A case study*, 23rd International Conference on Software Engineering (ICSE), 2001, pp. 555-564.

[IEEE90] IEEE, *Standard Glossary of Software Engineering, Terminology, IEEE Standard 610.12-1990*, 1990.

[IEEE98a] IEEE, *IEEE Recommended Practice for Software Requirements Specifications, IEEE Standard 830-1998*, 1998.

[IEEE98b] IEEE, *IEEE Guide for Information Technology - System Definition - Concept of Operations (ConOps) Document, IEEE Standard 1362-1998*, 1998.

[INC09] INCOSE, *INCOSE Requirements Management Tools Survey*, http://www.incose.org/ProductsPubs/Products/rmsurvey.aspx, last accessed 2009-12-29.

[ISO01] *ISO/IEC 9126-1:2001(E) : Software Engineering - Product Quality - Part 1: Quality Model*, International Organization for Standardization (ISO), 2001.

[ISO05] *ISO 25000:2005, Software Engineering -- Software product Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE*, International Organization for Standardization (ISO), 2005.

[ISO06] *ISO 9241-110:2006, Ergonomics of human-system interaction - Part 110: Dialogue principles*, International Organization for Standardization (ISO), 2006.

[ISO91] *ISO/IEC 9126:1991, Quality Characteristics and Guidelines for Their Use*, International Organization for Standardization (ISO), 1991.

[Jac99] S. Jacobs, *Introducing measurable quality requirements: a case study*, IEEE International Symposium on Requirements Engineering, 1999, pp. 172-179.

[JBR99] I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Software Development Process*, Addison Wesley, 1999.

[JP93] M. Jarke and K. Pohl, *Vision-Driven Requirements Engineering*, IFIP WG8.1 Working Conference on Information System Development Process, 1993, pp. 3-22.

[KBK+99] R. Kazman, M. Barbacci, M. Klein, S.J. Carriere and S.G. Woods, *Experience with Performing Architecture Tradeoff Analysis*, International Conference on Software Engineering, 1999.

[KD96] T.G. Kirner and A.M. Davis, *Nonfunctional Requirements of Real-Time Systems*, Advances in Computers, vol. 42, 1996, pp. 1-37.

[KDO07] M. Kassab, M. Daneva and O. Ormandjieva, *Scope Management of Non-Functional Requirements*, Euromicro Conference on Software Engineering and Advanced Applications, 2007.

[KDP+04] D. Kerkow, J. Dörr, B. Paech, T. Olsson and T. Koenig, *Elicitation and Documentation of Non-functional Requirements for Sociotechnical Systems*, Requirements Engineering for Sociotechnical Systems, A. S. José Luis Maté, ed., Idea Group, Inc., 2004.

[KHM+09] M. Klaes, J. Heidrich, J. Muench and A. Trendowicz, *CQML Scheme: A Classification Scheme for Comprehensive Quality Model Landscapes*, Proc. 35th EUROMICRO Conference Software Engineering and Advanced Applications, IEEE Computer Society, 2009, pp. 243-250.

[KKC00] R. Kazman, M. Klein and P. Clements, *ATAM: Method for Architecture Evaluation*, CMU/SEI-2000-TR-004, 2000.

[KKD03] D. Kerkow, K. Kohler and J. Doerr, *Usability and Other Quality Aspects Derived from Use Cases*, Performance by Design. Proceedings of forUSE 2003, 2003.

[KKP90] S.E. Keller, L.G. Kahn and R.B. Panara, *Specifying Software Quality Requirements with Metrics*, in Tutorial: System and Software Requirements Engineering, 1990, pp. 145-163.

[KOK04] H. Kaiya, A. Osada and K. Kaijiri, *Identifying stakeholders and their preferences about NFR by comparing use case diagrams of several existing systems*, 12th IEEE international Requirements Engineering Conference, 2004, pp. 112–121.

[KS95] G. Kotonya and I. Sommerville, *Integrationg Safety Analysis and Requirements Engineering*, CSEG/9/1995, Lancaster University, Computing Department, 1995.

[KS98] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*, John Wiley & Sons, 1998.

[LAG07] X. Liu, M. Azmoodeh and N. Georgalas, *Specification of Non-Functional Requirements for Contract Specification in the NGOSS Framework for Quality Management and Product Evaluation*, Workshop on Software Quality, 2007.

[Lam01a] A.v. Lamsweerde, *Goal-Oriented Requirements Engineering: A guided Tour*, 5th International Symposium on Requirements Engineering, 2001, pp. 249-261.

[Lam01b] A.v. Lamsweerde, *Building Formal Requirements Models for Reliable Software*, Lecture Notes in Computer Science No. 2043, Springer, 2001.

[LL98] A.v. Lamsweere and E. Letier, *Integrating Obstacles in Goal-Driven Requirements Engineering*, ICSE-98, 20th International Conference on Software Engineering, 1998.

[Loo03] L. Loomanns, *Essentials and Requisites for the Management of Evolution – Requirements and Incremental Validation*, Report No D1 part 3, ITEA EMPRESS Project, 2003.

[LWE01] B. Lawrence, K. Wiegers and C. Ebert, *The Top Risks of Requirements Engineering*, IEEE Software, vol. 18, no. 6, 2001.

[LX99] J. Lee and N.-L. Xue, *Analyzing User Requirements by Use Cases: A Goal-Driven Approach*, IEEE Software, vol. 16, no. 4, 1999.

[McC94] J. McCall, *Quality factors*, Encyclopedia of Software Engineering, 2, J. Marciniak, ed., John Wiley & Sons, 1994, pp. 958-969.

[MCY+92] J. Mylopulous, L. Chung, E. Yu and B. Nixon, *Representing and Using Non-Functional Requirements: A Process-Oriented Approach*, IEEE Transactions on Software Engineering, vol. 18, no. 6, 1992, pp. 483-497.

[Mil00] D. Miller, *Choice and Application of a Software Quality Model*, International Conference on Software Quality, 2000, pp. 243-252.

[MRS+07] S. Meier, T. Reinhard, R. Stoiber and M. Glinz, *Modeling and Evolving Crosscutting Concerns in ADORA*, Early Aspects at ICSE: Workshop in Aspect-Oriented Requirements Engineering and Architecture Design, 2007.

[MRS+09] C. Marhold, C. Rohleder, C. Salinesi and J. Doerr, *Clarifying Non-Functional Requirements to Improve User Acceptance - Experience at Siemens*, Requirements Engineering: Foundation for Software Quality (REFSQ): 15th International Working Conference, 2009 pp. 139-146.

[MS95] D.N.J. Mostert and S.H.v. Solms, *A Technique to Include Computer Security, Safety, and Resilience Requirements as Part of the Requirements Specification*, Journal of Systems Software, vol. 31, 1995, pp. 45-53.

[MYB+91] A. Maclean, R.M. Young, V.M.E. Belotti and T.P. Moran, *Questions, Options and Criteria: Elements of design space analysis*, Human Computer Interaction - Special Issue on Design Rationale, vol. 6, no. 3 & 4, 1991.

[Nix00] B. Nixon, *Management of Performance Requirements for Information Systems*, IEEE Transactions on Software Engineering, vol. 26, no. 12, 2000.

[OMG05] Object Management Group, *UML Profile for Schedulability, Performance, and Time Specification. Version 1.1*, 2005.

[Pas03] T. Pasternak, *Using Trade-Off Analysis to Uncover Links between Functional and Non-Functional Requirements in Use-Case Analysis*, IEEE International Conference on Software - Science, Technology & Engineering, 2003.

[PK04a] B. Paech and D. Kerkow, *Non-functional requirements engineering – quality is essential*, 10th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ), 2004.

[PK04b] B. Paech and K. Kohler, *Task-Driven Requirements in Object-Oriented Development*, Perspectives on Software Requirements, The Kluwer Interna-

tional Series in Engineering and Computer Science SECS 753, J. C. S. d. P. Leite and J. H. Doorn, eds., Kluwer Academic Publishers, 2004, pp. 45-67.

[Pls87] P.E. Plsek, *Defining Quality at the Marketing/Development Interface*, Proc. Quality Progress, June 1987, pp. 28-36.

[Poh08] K. Pohl, *Requirements Engineering: Grundlagen, Prinzipien, Techniken* Dpunkt.Verlag GmbH, 2008

[PvKD+03] B. Paech, A. von Knethen, J. Doerr, J. Bayer, D. Kerkow, R. Kolb, A. Trendowicz, T. Punter and A. Dutoit, *An experience based approach for integrating architecture and requirements engineering*, From Software Requirements to Architectures (STRAW), Workshop held at ICSE, 2003.

[Qua09] *Quamoco Project Website*, http://quamoco.in.tum.de/, last accessed 2009-11-21.

[RJB99] J. Rumbaugh, I. Jacobson and G. Booch, *The Unified Modeling Lanuage Reference Manual*, Addison-Wesley Professional, 1999.

[Rob89] W.N. Robinson, *Integrating Multiple Specifications Using Domain Goals*, IWSSD-5, 5th International Workshop on Software Specification and Design, 1989, pp. 219-225.

[Rom08] H.D. Rombach and F. Seelisch, *Formalisms in Software Engineering: Myths Versus Empirical Facts*, Central and East European Conference on Software Engineering Techniques CEE-SET 2007 - Revised Selected Papers, 2008.

[Rom09] D. Rombach, *Lecture "Grundlagen des Software Engineering", Chapter 4*, 2009, http://wwwagse.informatik.uni-kl.de/teaching/gse/ws2009, last accessed 2009-12-29.

[Rom85] G.-C. Roman, *A Taxonomy of Current Issues in Requirements Engineering*, IEEE Computer, 1985, pp. 14-22.

[RR99] S. Robertson and J. Robertson, *Mastering the Requirements Process*, ACM Press, 1999.

[Rup07] C. Rupp, *Requirements-Engineering und -Management (4. Auflage)*, Hanser Fachbuchverlag, 2007.

[RW07] T. Rinke and T. Weyer, *Defining Reference Models for Modelling Qualities: How Requirements Engineering Techniques Can Help*, International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ), 2007.

[Sch99] A.-W. Scheer, *ARIS - Business Process Frameworks (3rd Edition)*, Springer, 1999.

[SHR09] X. Song, B. Hwong and J. Ros, *Experiences in Developing Quantifiable NFRs for the Service-Oriented Software Platform*, 17th IEEE International Requirements Engineering Conference, 2009.

[SM98] A. Sutcliffe and S. Minocha, *Scenario-based Analysis of Non-Functional Requirements*, Requirements Engineering: Foundation for Software Quality (REFSQ), Workshop held at CAiSE, 1998.

[URW+08] O. Uenalan, N. Riegel, S. Weber and J. Doerr, *Using Enhanced Wiki-based Solutions for Managing Requirements*, MARK '08: First International Workshop on Managing Requirements Knowledge, Workshop held at International Conference on Requirements Engineering, 2008, pp. 63-67.

[VDG08] K.B. Villela, J. Doerr and A. Groß, *Proactively Managing the Evolution of Embedded System Requirements* IEEE Computer Society: 16th IEEE International Requirements Engineering Conference, 2008, pp. 13-22.

[vSB99] R.v. Solingen and E. Berghout, *The Goal/Question/Metric Method: A practical guide for quality improvement of software development*, McGraw-Hill, 1999.

[WDW08] S. Wagner, F. Deissenboeck and S. Winter, *Erfassung, Strukturierung und Überprüfung von Qualitätsanforderungen durch aktivitätenbasierte Qualitätsmodelle*, 2. Workshop zur Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung, Workshop held at Software Engineering, 2008.

[Wie03] K. Wiegers, *Software Requirements*, Microsoft Press, 2003.

[Wor07] J. Dörr and P. Liggesmeyer, *Workshop zur Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung*, Proc. Software Engineering 2007.

[Wor08] J. Dörr and P. Liggesmeyer, *2. Workshop zur Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung*, Proc. Software Engineering, 2008.

[Wor10] J. Dörr and P. Liggesmeyer, *3. Workshop zur Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung*, Proc. Software Engineering, 2010, to appear.

[WRH+00] C. Wohlin, P. Runeson, M. Höst, M.Ohlsson, B. Regnell and A. Wesslén, *Experimentation in Software Engineering – An Introduction*, Kluwer Academic Publisher, 2000.

[Yu93] E.S.K. Yu, *Modelling Organizations for Information Systems Requirements Engineering*, RE'93 - 1st International Symposium on Requirements Engineering, 1993, pp. 34-41.

[Yu97] E.S.K. Yu, *Towards Modeling and Reasoning Support for Early-Phase requirements Engineering*, 3rd IEEE Int. Symp. on Requirements Engineering 1997.

[Yue87] K. Yue, *What does it mean to say that a specification is complete?*, IWSSD-4, Fourth International Workshop on Software Specification and Design, 1987.

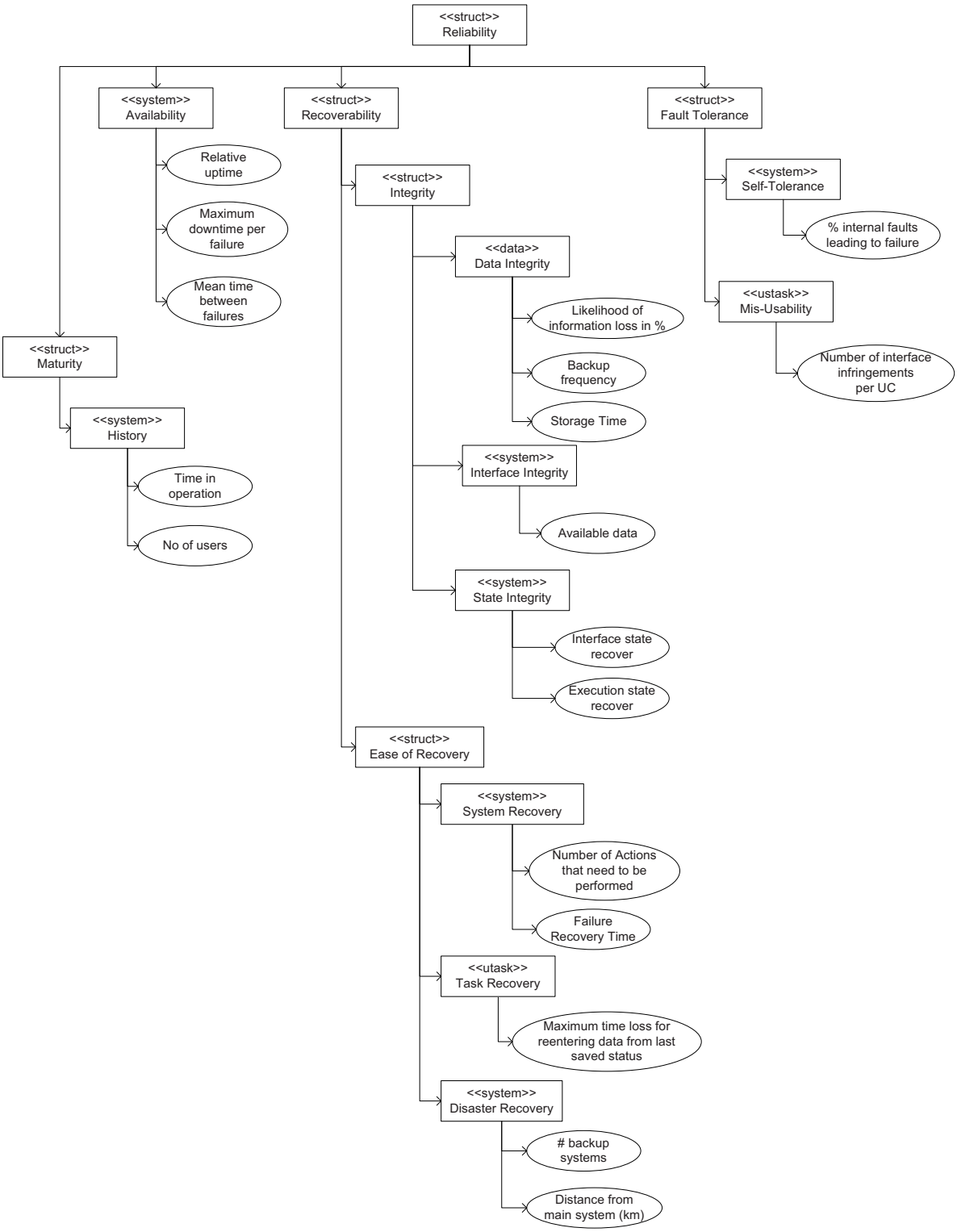
[YZC+84] R. Yeh, P. Zave, A. Conn and G. Cole, *Software Requirements: New Directions and Perspectives*, Handbook of Software Engineering, 1984, pp. 519-543.

[Zav97] P. Zave, *Classification of Research Efforts in Requirements Engineering*, ACM Computing Surveys, vol. 29, no. 4, 1997, pp. 315-321.

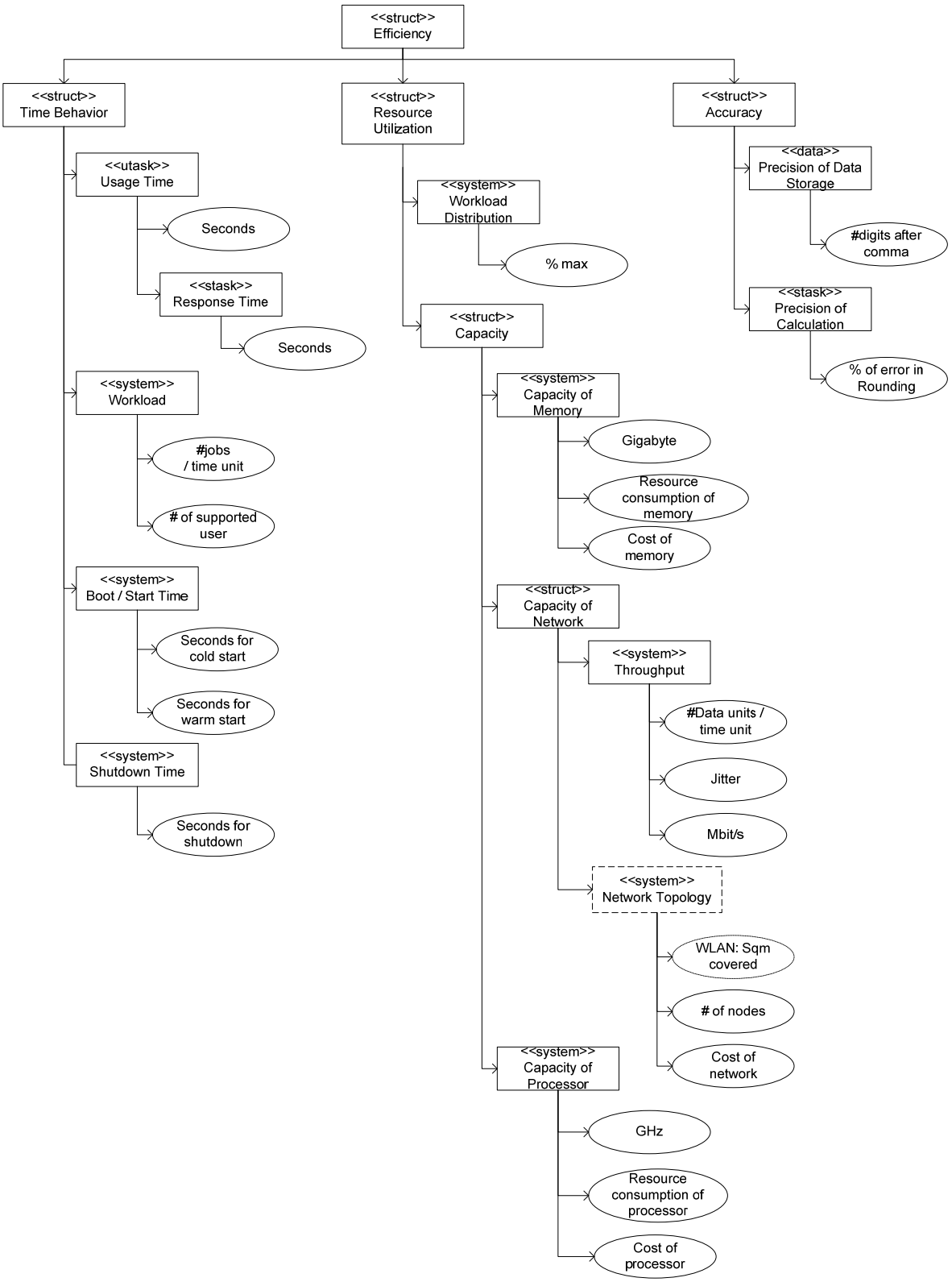
Appendix A: Example Quality Models

In this section of the appendix, some excerpts from experience-based reference quality models for the typical elements of HLQA, i.e., the QAs reliability, efficiency, security, usability, portability, and maintainability are presented. The purpose of these quality models is to give the reader an illustration on how reference quality models can look like. None of them claims to be a complete reference quality model. They shall be viewed as simple examples that should be easy to understand. For this purpose, most of the QAs in the following quality models should be self-descriptive. A precise definition of the QAs and metrics is needed if the models shall be used as reference quality models for an organization. The terminology for the QAs must be agreed upon by the stakeholders in the tailoring workshop (see process activity P1.2). For an initial terminology of the QAs, the definitions of the QAs from standards such as ISO 9126, Part 1 [ISO01] and ISO 9241, Part 110 [ISO06] can be taken as starting point (most of the QAs in the example models can be found there).

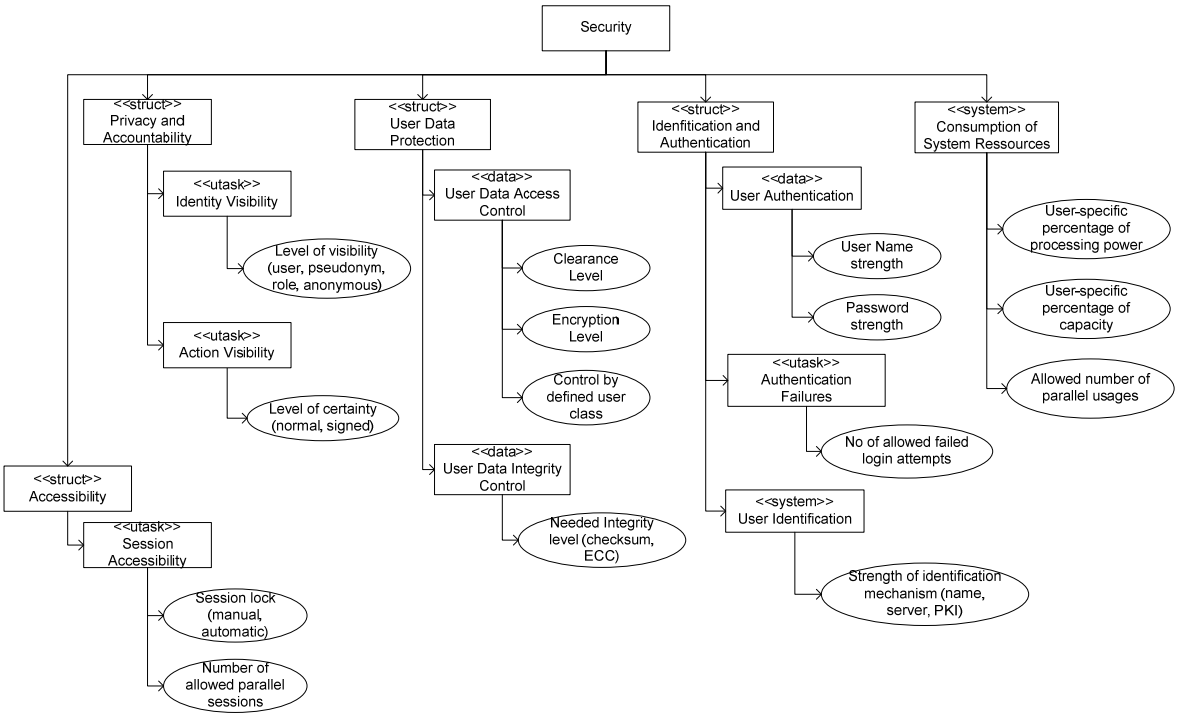
A.1: Reliability



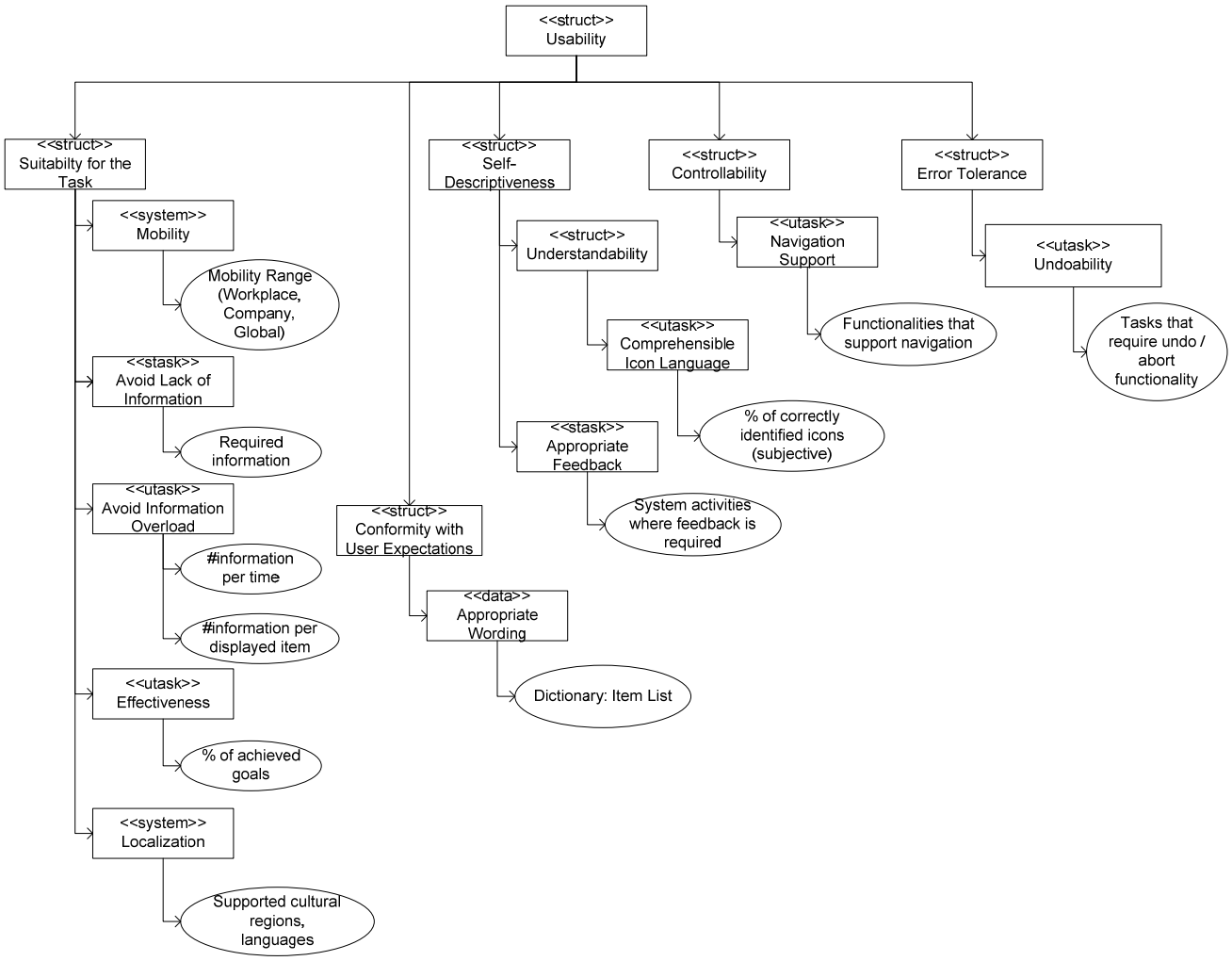
A.2: Efficiency



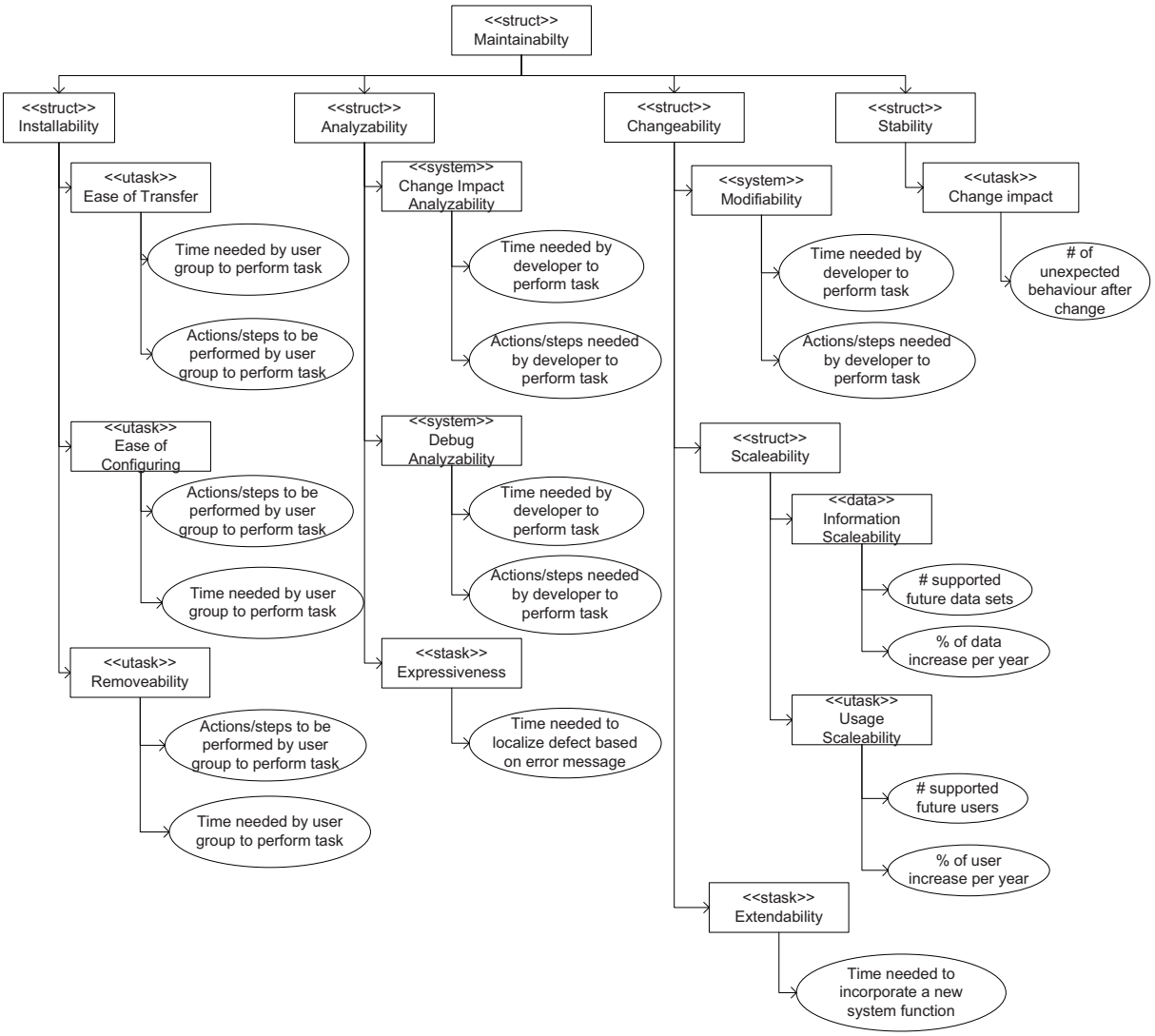
A.3: Security


Security
<<struct>> Privacy and Accountability
<<utask>> Identity Visibility
Level of visibility (user, pseudonym, role, anonymous)
<<utask>> Action Visibility
Level of certainty (normal, signed)
<<struct>> Accessibility
<<utask>> Session Accessibility
Session lock (manual, automatic)
Number of allowed parallel sessions
<<struct>> User Data Protection
<<data>> User Data Access Control
Clearance Level
Encryption Level
Control by defined user class
<<data>> User Data Integrity Control
Needed Integrity level (checksum, ECC)
<<struct>> Idenfitication and Authentication
<<data>> User Authentication
User Name strength
Password strength
<<utask>> Authentication Failures
No of allowed failed login attempts
<<system>> User Identification
Strength of identification mechanism (name, server, PKI)
<<system>> Consumption of System Ressources
User-specific percentage of processing power
User-specific percentage of capacity
Allowed number of parallel usages

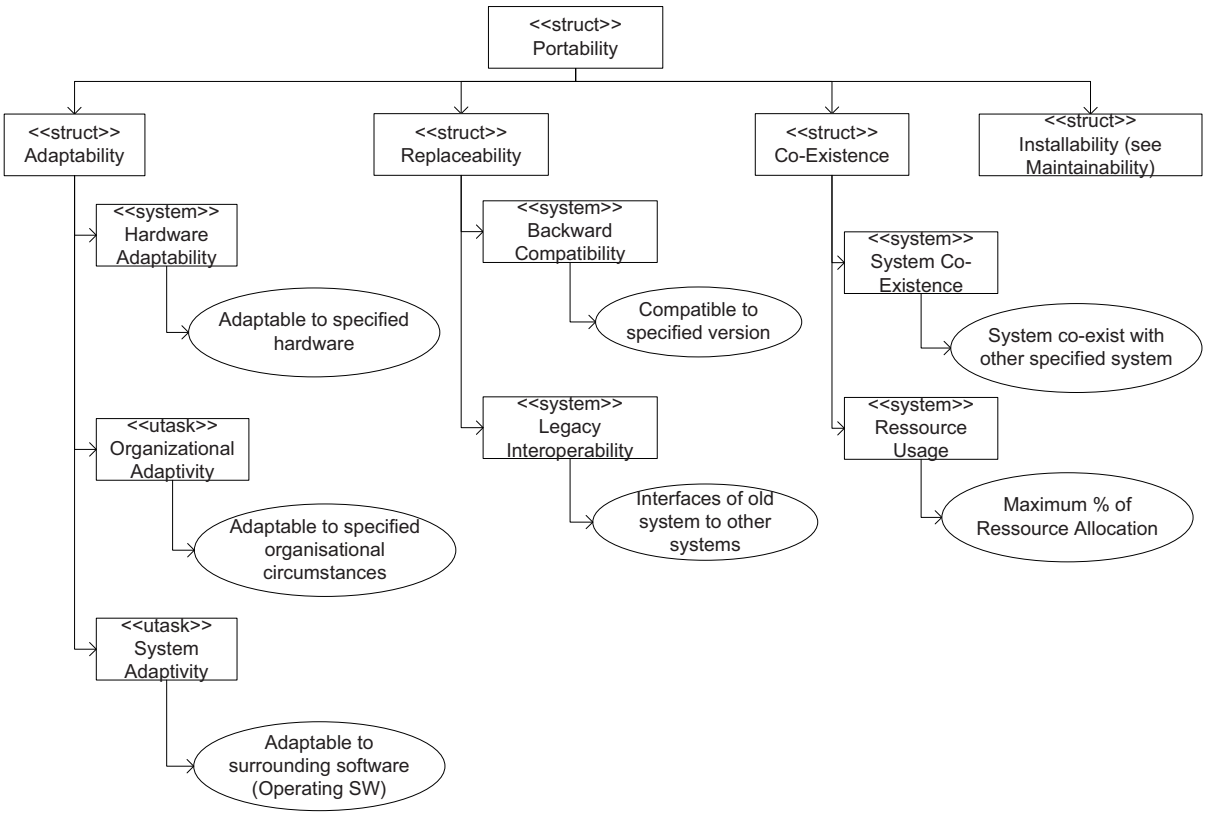
A.4: Usability



A.5: Maintainability



A.6: Portability



Appendix B: Template for an Integrated Specification

Requirements Specification for

<System name>

<System-ID>

Document-Version <Number>
Date<Date>

Distribution List
<Names>

Contact Person

Document history

Date	*Person*	*Change*	*Reason*	*Review date*

TOC

1. Introduction
    1.1 System Identification
    1.2 System Overview
    1.3 Document Overview
2. Business Processes
3. Operational Scenarios
    3.1 Overview on Operational Scenarios
    3.2 Description of Operational Scenarios
4. System Function Descriptions
5. Data Model
6. General Non-Functional Requirements
7. First Solution Ideas

Glossary

Term/Abbreviation	*Description*

Reference documents

Reference ID	*Description and Link*

1. Introduction

1.1 System Identification

Name, Number and Abbreviation for the SUD.

1.2 System Overview

Rough description of the system. Overview on physical architecture (set of subsystems).

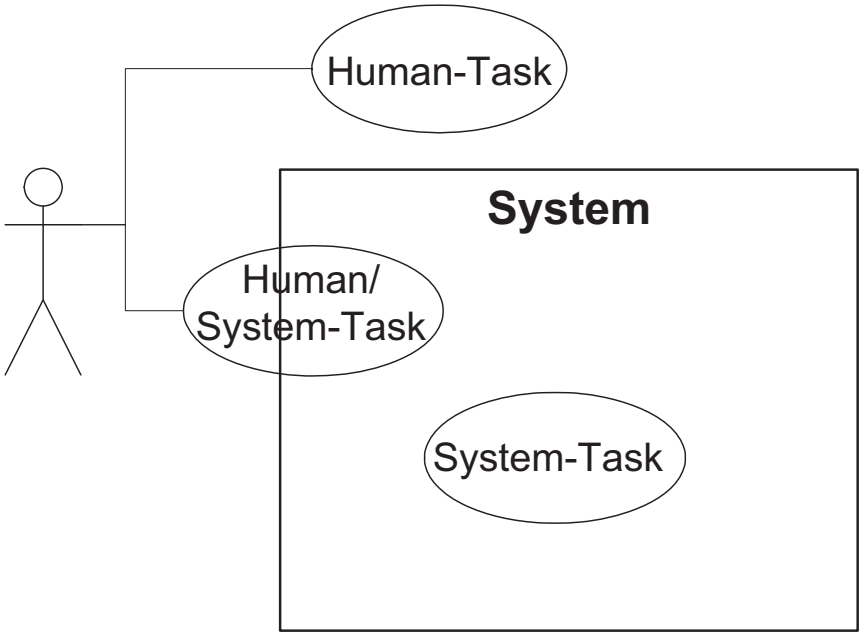
1.3 Document Overview

2. Business Processes

BP <number>	
Name	
Goal	
Precondition	
Trigger	
Possibilities to take influence	
Priority	
Input	
Output	
Postcondition	
Execution Profile (Complexity, Rate)	
NFR	
Ressources	
To-Be Workflow *(EPC)*	
Textual description of workflow	

3. Operational Scenarios

3.1 Overview on Operational Scenarios



User Task QA1: NFR1
User Task QA2: NFR3

3.2 Description of Operational Scenarios

UC<number>	
Name	
Goal	
Actor	
Precondition	
Flow of Events	
Exceptions	
Rules	
NFR	
Data	
System Functions	
Postcondition	

4. System Function Descriptions

SF<number>	
Name	
Input	
Precondition	
Description	
Exceptions	
Rules	
NFR	
Output	
Involved Components	
Post Condition	

5. Data Model

Data Item<number>	
Name	
Description	
Quantity	
Dependencies	
NFR	

6. General Non-Functional Requirements

6.1 HLQA1

6.1.1 Overall System

System-QA1:
Task-Overspanning-QA1:

6.1.2 Subsystem 1

6.1.3 Subsystem 2

6.1.4 …

6.2 HLQA2

6.3 *…*

7. First Solution Ideas

List of all means that are already determined.

Appendix C: Effort Data for Case Studies

Case Study / Activity	#Method Experts	#Domain Experts	Duration (minutes)	SUM all method experts per activity (hours)	SUM all domain experts per activity (hours)	SUM all persons per activity (hours)	SUM all method experts all activities (days)	SUM all domain experts all activities (days)	SUM all persons all activity (days)	SUM per domain expert all activities (days)	Automation Potential by Checklist Generation Tool (days)	Automation Potential by Elicitation Guide Tool (days)
Empress												
Prioritization	2	1	30	1	0,5	1,5						
Preparation Efficiency	2	0	240	8	0	8						
Eff.-Tailoring	2	1	180	6	3	9						
Eff.-Elicitation	2	1	120	4	2	6						
Preparation Maint.+Rel.	2	0	240	8	0	8						
Maintainability -Tailoring	2	2	60	2	2	4						
Maintainability -Elicitation	2	2	120	4	4	8						
Reliability - Tailoring	2	2	60	2	2	4						
Reliability - Elicitation	2	2	120	4	4	8						
SUM all activities				**39**	**17,5**	**56,5**	**5**	**2**	**7**	**1,44**	**2**	**1**
Ricoh												
Preparation	1	0	60	1	0	1						
Tailoring	2	2	240	8	8	16						
Elicitation	2	2	480	16	16	32						
SUM all activities				**25**	**24**	**49**	**3,13**	**3**	**6**	**1,5**	**0,13**	**2**
GIS												
No data available												
FIN												
Preparation	2	0	60	2	0	2						
Tailoring	2	8	480	16	64	80						
Checklist Derivation	1	0	120	2	0	2						
Elicitation	2	8	960	32	128	160						
SUM all activities				**52**	**192**	**244**	**6,5**	**24**	**31**	**3**	**0,25**	**4**
MBTech												
Checklist Derivation	1	0	145	2,42	0	2,42						
Tailoring	2	4	720	24	48	72						
Elicitation	2	4	720	24	48	72						
SUM all activities				**50,42**	**96**	**146,42**	**6,3**	**12**	**18,3**	**3**	**0,3**	**6**
SOL												
No data available												
EMERGE												
Preparation	2	0	420	14	0	14						
Workshop Prioritization / Tailoring 1	1	7	30	0,5	3,5	4						
Workshop Prioritization / Tailoring 2	1	3	20	0,33	1	1,33						
Derive Qualitymodels & generate checklists	1	0	600	10	0	10						
Workshop Elicitation 1	1	3	45	0,75	2,25	3						
Workshop Elicitation 2	1	3	60	1	3	4						
Specification of NFRs into Integrated Specification	1	0	600	10	0	10						
SUM all activities				**36,58**	**9,75**	**46,33**	**4,57**	**1**	**6**	**0,19**	**N/A**	**0,22**
ES												
Preparation	2	0	240	8	0	8						
Tailoring	2	5	480	16	40	56						
SUM all activities				**24**	**40**	**64**	**3**	**5**	**8**	**1**	**N/A**	**N/A**

Appendix D: Detailed Expert Estimate in EMERGE

1st Level QA	2nd Level QA	3rd Level QA	Reference to NFRs in document (page number)	NFRs of this kind completely fulfilled in current system (yes/no)	Estimated Effort to completely fulfill the NFR on granularity 2nd level (hours)
Efficiency	Time Behaviour	Response Time Transmission Time Workload	34, 35, 36, 93, 94, 96, 100, 107, 109, 118	yes	
	Resource Utilisation	Workload Distribution Processing capacity Information distribution	34, 35, 36, 96, 98, 107	yes	
	Usage Time	Definition of approporiate time-outs	34, 35, 36, 96, 100	yes	
Reliability	Maturity	History Failure resolution Fault removal Mean time between failures	38, 39, 40	no	160h
	Recoverability	Integrity Ease of recovery Last saved status	38, 39, 40, 96, 98, 107	no	120h
Safety	Risks	Risk Handling	51	no	80h
	Accidents	Accident Handling	51	yes	
Usability	Suitability for the task	Mobility Avoid information overload Avoid lack of information Effectiveness Localisation	44, 46, 48, 96, 98, 107, 118, 120	yes	
	Self-Descriptiveness	Understandability Appropriate feedback of system activities Comprehensible Icon Language	44, 46,48, 94, 99, 100, 109	yes	
	Controllability	Traceability Navigation support	44, 46, 48, 96, 98, 107, 119, 120	yes	
	Conformity with user expectations	Consistency Appropriate wording for each target group	44, 46, 48, 96, 98, 107, 119, 120	yes	
	Error Tolerance	Undo, abort, feasability of medical parameters	44, 46, 100	yes	
	Suitability for Individualization	Accessibility Adaptability Customizability	46	no	120h
	Suitability for Learning	Experience Availability of a help system	44, 46, 48	no	80h
	Acceptance	Relevant functionalities Costs Perceived usefulness Unobtrusiveness of system components	44, 46, 48	yes	
	Attractiveness	Satisfaction Device form factor	46,48	no	160h
Security	Vulnerability		49	no	80h
	Privacy		49	no	20h
	Encryption		49	yes	
	Access / Authorization		49	yes	
				SUM in hours:	820h
				SUM in days:	102,5 d

Appendix E: List of Abbreviations

EPC	Event-driven process chain
FR	Set of all functional requirements including user task, system task and data item requirements
HLQA	Set of high-level quality attributes
ISO	International Standardization Organization
NFR	Non-functional requirement
QA	Quality attribute
SUD	System under development
SYS	Set of all subsystems and the system itself.
TORE	Task- and object-oriented requirements engineering
UC	Use Case

Lebenslauf

Persönliche Daten

Name	Jörg Dörr
Anschrift	Hainbuchenweg 12 67661 Kaiserslautern
Geburtsdatum und -ort	05.03.1976 in Kaiserslautern
Familienstand	Verheiratet, 2 Kinder

Werdegang

1982 – 1984	Pestalozzi-Grundschule, Kaiserslautern
1984 – 1986	Stresemann-Grundschule Kaiserslautern
1986 – 1995	Hohenstaufen-Gymnasium, Kaiserslautern (Abitur)
1995 – 1996	Zivildienst bei der Lebenshilfe e.V., Kaiserslautern
1996 – 2002	Studium der Informatik, Universität Kaiserslautern (Diplom)
Seit 2002	Wissenschaftlicher Mitarbeiter am Fraunhofer-Institut für Experimentelles Software Engineering (IESE), Kaiserslautern
2004 – 2005	Leiter des Geschäftsfelds „Telekommunikation und Dienstanbieter“ am Fraunhofer IESE, Kaiserslautern
2004 – 2006	Leiter der Abteilung „PR/Marketing“ am Fraunhofer IESE, Kaiserslautern
2005 – 2009	Leiter der Abteilung „Requirements and Usability Engineering“ am Fraunhofer IESE, Kaiserslautern
Seit 2010	Leiter der Abteilung „Information & Interactive Systems“ am Fraunhofer IESE, Kaiserslautern
Seit Juni 2010	Leiter der Hauptabteilung „Information Systems“ am Fraunhofer IESE, Kaiserslautern

Kaiserslautern, den 7.Mai 2011

PhD Theses in Experimental Software Engineering

Volume 1 **Oliver Laitenberger** (2000), *Cost-Effective Detection of Software Defects Through Perspective-based Inspections*

Volume 2 **Christian Bunse** (2000), *Pattern-Based Refinement and Translation of Object-Oriented Models to Code*

Volume 3 **Andreas Birk** (2000), *A Knowledge Management Infrastructure for Systematic Improvement in Software Engineering*

Volume 4 **Carsten Tautz** (2000), *Customizing Software Engineering Experience Management Systems to Organizational Needs*

Volume 5 **Erik Kamsties** (2001), *Surfacing Ambiguity in Natural Language Requirements*

Volume 6 **Christiane Differding** (2001), *Adaptive Measurement Plans for Software Development*

Volume 7 **Isabella Wieczorek** (2001), *Improved Software Cost Estimation A Robust and Interpretable Modeling Method and a Comprehensive Empirical Investigation*

Volume 8 **Dietmar Pfahl** (2001), *An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisations*

Volume 9 **Antje von Knethen** (2001), *Change-Oriented Requirements Traceability Support for Evolution of Embedded Systems*

Volume 10 **Jürgen Münch** (2001), *Muster-basierte Erstellung von Software-Projektplänen*

Volume 11 **Dirk Muthig** (2002), *A Light-weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines*

Volume 12 **Klaus Schmid** (2003), *Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines*

Volume 13 **Jörg Zettel** (2003), *Anpassbare Methodenassistenz in CASE-Werkzeugen*

Volume 14 **Ulrike Becker-Kornstaedt** (2004), *Prospect: a Method for Systematic Elicitation of Software Processes*

Volume 15 **Joachim Bayer** (2004), *View-Based Software Documentation*

Volume 16 **Markus Nick** (2005), *Experience Maintenance through Closed-Loop Feedback*

Volume 17 **Jean-François Girard** (2005), *ADORE-AR: Software Architecture Reconstruction with Partitioning and Clustering*

Volume 18 **Ramin Tavakoli Kolagari** (2006), *Requirements Engineering für Software-Produktlinien eingebetteter, technischer Systeme*

Volume 19 **Dirk Hamann** (2006), *Towards an Integrated Approach for Software Process Improvement: Combining Software Process Assessment and Software Process Modeling*

Volume 20 **Bernd Freimut** (2006), *MAGIC: A Hybrid Modeling Approach for Optimizing Inspection Cost-Effectiveness*

Volume 21 **Mark Müller** (2006), *Analyzing Software Quality Assurance Strategies through Simulation. Development and Empirical Validation of a Simulation Model in an Industrial Software Product Line Organization*

Volume 22 **Holger Diekmann** (2008), *Software Resource Consumption Engineering for Mass Produced Embedded System Families*

Volume 23 **Adam Trendowicz** (2008), *Software Effort Estimation with Well-Founded Causal Models*

Volume 24 **Jens Heidrich** (2008), *Goal-oriented Quantitative Software Project Control*

Volume 25 **Alexis Ocampo** (2008), *The REMIS Approach to Rationale-based Support for Process Model Evolution*

Volume 26 **Marcus Trapp** (2008), *Generating User Interfaces for Ambient Intelligence Systems; Introducing Client Types as Adaptation Factor*

Volume 27 **Christian Denger** (2009), *SafeSpection – A Framework for Systematization and Customization of Software Hazard Identification by Applying Inspection Concepts*

Volume 28 **Andreas Jedlitschka** (2009), *An Empirical Model of Software Managers' Information Needs for Software Engineering Technology Selection A Framework to Support Experimentally-based Software Engineering Technology Selection*

Volume 29 **Eric Ras** (2009), *Learning Spaces: Automatic Context-Aware Enrichment of Software Engineering Experience*

Volume 30 **Isabel John** (2009), *Pattern-based Documentation Analysis for Software Product Lines*

Volume 31 **Martín Soto** (2009), *The DeltaProcess Approach to Systematic Software Process Change Management*

Volume 32 **Ove Armbrust** (2010), *The SCOPE Approach for Scoping Software Processes*

Volume 33 **Thorsten Keuler** (2010), *An Aspect-Oriented Approach for Improving Architecture Design Efficiency*

Volume 34 **Jörg Dörr** (2010), *Elicitation of a Complete Set of Non-Functional Requirements*

Software Engineering has become one of the major foci of Computer Science research in Kaiserslautern, Germany. Both the University of Kaiserslautern's Computer Science Department and the Fraunhofer Institute for Experimental Software Engineering (IESE) conduct research that subscribes to the development of complex software applications based on engineering principles. This requires system and process models for managing complexity, methods and techniques for ensuring product and process quality, and scalable formal methods for modeling and simulating system behavior. To understand the potential and limitations of these technologies, experiments need to be conducted for quantitative and qualitative evaluation and improvement. This line of software engineering research, which is based on the experimental scientific paradigm, is referred to as 'Experimental Software Engineering'.
In this series, we publish PhD theses from the Fraunhofer Institute for Experimental Software Engineering (IESE) and from the Software Engineering Research Groups of the Computer Science Department at the University of Kaiserslautern. PhD theses that originate elsewhere can be included, if accepted by the Editorial Board.

ISBN 978-3-8396-0261-4

Fraunhofer IESE

Technische Universität Kaiserslautern
AG Software Engineering

PhD Theses in Experimental Software Engineering