Are “Non-functional” Requirements really Non-functional?

An Investigation of Non-functional Requirements in Practice

Jonas Eckhardt, Andreas Vogelsang, and Daniel Méndez Fernández
Technische Universität München, Germany
{eckharjo|vogelsan|mendezfe}@in.tum.de

ABSTRACT
Non-functional requirements (NFRs) are commonly distinguished from functional requirements by differentiating how the system shall do something in contrast to what the system shall do. This distinction is not only prevalent in research, but also influences how requirements are handled in practice. NFRs are usually documented separately from functional requirements, without quantitative measures, and with relatively vague descriptions. As a result, they remain difficult to analyze and test. Several authors argue, however, that many so-called NFRs actually describe behavioral properties and may be treated the same way as functional requirements. In this paper, we empirically investigate this point of view and aim to increase our understanding on the nature of NFRs addressing system properties. We report on the classification of 530 NFRs extracted from 11 industrial requirements specifications and analyze to which extent these NFRs describe system behavior. Our results suggest that most “non-functional” requirements are not non-functional as they describe behavior of a system. Consequently, we argue that many so-called NFRs can be handled similarly to functional requirements.

Categories and Subject Descriptors
D.2.1 [Software Engineering]: Requirements/Specifications

Keywords
Non-functional requirements, classification, model-based development, empirical studies

1. INTRODUCTION

One conventional distinction between non-functional requirements (NFRs) and functional requirements is made by differentiating how the system shall do something in contrast to what the system shall do [24, 25]. This distinction is not only prevalent in research, but it also influences how requirements are elicited, documented, and validated in practice [1, 3, 9, 26]. As a matter of fact, up until now there does not exist a commonly accepted approach for the NFR-specific elicitation, documentation, and analysis [3, 26]; NFRs are usually described vaguely [3, 1], remain often not quantified [26], and as a result remain difficult to analyze and test [1, 3, 26]. Furthermore, NFRs are often retrofitted in the development process or pursued in parallel with, but separately from, functional requirements [9] and, thus, are implicitly managed with little or no consequence analysis [26]. This limited focus on NFRs can result in the long run in high maintenance costs [26].

Although the importance of NFRs for software and systems development is widely accepted, the discourse about how to handle NFRs is still dominated by how to differentiate them exactly from functional requirements [6, 14]. One point of view is that the distinction is an artificial one and we should rather differentiate between behavior (e.g., response times) and representation (e.g., programming languages). The underlying argument is that most NFRs actually describe behavioral properties [14] and should be treated the same way as functional requirements in the software development process [5]. Behavioral properties subsume classical functional requirements, such as “the user must be able to remove articles from the shopping basket” as well as NFRs which describe behavior such as “the system must react on every input within 10ms”. Representational properties include NFRs that determine how a system shall be syntactically or technically represented, such as “the software must be implemented in the programming language Java” [5, 6].

In this paper, we empirically investigate this point of view and aim to increase our understanding on the nature of NFRs addressing system properties. To this end, we classify 530 NFRs extracted from 11 industrial requirements specifications with respect to their kind. Our results show that 75% of the requirements labeled as “non-functional” in the considered industrial specifications describe system behavior and only 25% describe the representation of the system. As behavior has many facets, we further classify behavioral NFRs according to the system view they address (interface, architecture, or state), and the behavior theory used to express them (syntactic, logical, probabilistic, or timed) [5, 6]. Based on this fine-grained classification, we discuss the implications we see on handling NFRs in the software engineering disciplines, e.g., testing or design.

Based on the results of our study, we conclude that most “non-functional” requirements are misleadingly declared as such because they actually describe behavior of the system. This in turn means that many so-called NFRs can be handled
similarly to functional requirements. Please note that the focus of this paper is not to criticize the term “non-functional” but to expose the artificial separation of functional and non-functional (or quality) requirements in practice.

The remainder of the paper is structured as follows: In Section 2, we discuss background and related work, and, subsequently, we present our study design in Section 3. We report on the results in Section 4 and discuss the threats to validity and our mitigation strategies in Section 5. In Section 6, we provide a discussion of the overall results and their impact on theory and practice, before concluding our work in Section 7.

2. BACKGROUND & RELATED WORK

In this section, we provide background and related work on NFR classifications and on the implications of NFRs on software development.

Previously published material. In our previously published paper [12], we presented a research proposal with the goal of analyzing natural language NFRs taken from industrial requirements specifications to better understand their nature. Our study reported here, relies on and extends our previous study design. We present the results in full detail, and provide a comprehensive discussion on the implications on software engineering disciplines.

NFR classifications. There exist several classification schemes for NFRs in literature (e.g., [8, 14, 16, 23, 25, 29]). One example for such a classification, which is based on a quality model, is the ISO/IEC 9126 [16]. It defines external and internal quality of a software system and derives several quality characteristics (e.g., Functionality–Security or Portability–Installability). Sommerville further provides a classification scheme based on a distinction between process requirements, product requirements, and external requirements [25]. We base our distinction of NFR classes on the ISO/IEC 9126 classification. Furthermore, we exclude process requirements from our study, as they do not describe properties of the system itself.

Pohl [23] discusses the misleading use of the term “non-functional” and argues to use “quality requirements” for product-related NFRs that are not constraints. Glinz [14] performs a comprehensive review on the existing definitions of NFRs, analyzes problems with these definitions, and proposes a definition on his own. He highlights three different problems with the current definitions: a definition problem, i.e., NFR definitions have discrepancies in the used terminology and concepts, a classification problem, i.e., the definitions provide very different sub-classifications of NFRs, and finally a representation problem, i.e., the notion of NFRs is representation-dependent. In our study, we faced all of the three problems: we motivate our study based on the definition and classification problem and during the execution of our study, we faced the representation problem (see also our discussion on threats to validity in Section 5). Although we agree on the critique about the obsolete and misleading notion of the term “non-functional”, it still dominates the way requirements are handled in practice, as reflected in our data.

Mairiza et al. [19] perform a literature review on NFRs, investigating the notion of NFRs in the software engineering literature to increase the understanding of this complex and multifaceted phenomenon. Amongst others, they found about 114 different NFR classes. As a result of a frequency analysis, they found that the five most frequently mentioned NFR classes in literature are performance, reliability, usability, security, and maintainability (in that order). In our study, we got similar results: we found that the five most frequently used NFR classes in our industrial specifications are security, reliability, usability, efficiency, and portability (in that order). While Mairiza et al. performed their analysis on available literature, our study analyzes NFRs documented in industrial projects.

NFRs and their implications on software development. One of the first studies that analyzed how to systematically deal with NFRs in software development was conducted by Chung and Nixon [9]. They argue that NFRs are often retrofitted in the development process or pursued in parallel with, but separately from, functional design and that an ad-hoc development process often makes it hard to detect defects early. They perform three experimental studies on how well a given framework [22] can be used to systematically deal with NFRs. Svensson et al. [26] perform an interview study on how quality requirements are used in practice. Based on their interviews, they found that there is no NFR-specific elicitation, documentation, and analysis, that NFRs are often not quantified and, thus, difficult to test, and that there is only an implicit management of NFRs with little or no consequence analysis. Furthermore, they found that at the project level, NFRs are not taken into consideration during product planning (and are thereby not included as hard requirements in the projects) and they conclude that the realization of NFRs is a reactive rather than proactive effort. Borg et al. [3] analyze via interviews how NFRs are handled in practice by the example of two Swedish software development organizations. They found that NFRs are difficult to elicit because of a focus on functional requirements, they are often described vaguely, are often not sufficiently considered and prioritized, and they are sometimes even ignored. Furthermore, they state that most types of NFRs are difficult to test properly due to their nature, and when expressed in non-measurable terms, testing is time-consuming or even impossible. Ameller et al. [1] perform an empirical study based on interviews around the question How do software architects deal with NFRs in practice? They found that NFRs were not often documented, and even when documented, the documentation was not always precise and usually became desynchronized. Furthermore, they state that NFRs were claimed to be mostly satisfied at the end of the project although just a few classes were validated. With respect to model-driven development, Ameller et al. [2] show that most model-driven development (MDD) approaches focus only on functional requirements and do not integrate NFRs into the MDD process. They further identify challenges to overcome in order to integrate NFRs in the MDD process effectively. Their challenges include modeling of NFRs at the PIM-level, which includes the question which types of NFRs are most relevant to the MDD process? According to Ameller et al. [2], the few MDD approaches that support the modeling of NFRs can be classified into approaches that use UML extensions [13, 28, 31] or a specific metamodel [15, 18, 21] to model NFRs. In all of the approaches, functional requirements and NFRs are modeled separately. Damm et al. [11] suggest to overcome this separation and propose a so-called rich component model based
3. STUDY DESIGN

In this section, we describe our overall goal, our research questions, and the design of our study.

3.1 Goal and Research Questions

The goal of this study is to increase our understanding on the nature of NFRs addressing system properties. In particular, we are interested in understanding to which extent these NFRs and their respective classes (e.g., security or reliability) describe system behavior and what kind of behavior they address. This allows us to discuss the implications on handling NFRs in the software engineering disciplines (e.g., testing or design).

To achieve our goal, we formulate the following research questions (RQs), which we cluster in two categories:

1. Distribution of NFR classes in practice.

We examine the distribution of NFR classes in practice via two research questions:

**RQ1** What NFR classes are documented in practice? With this RQ, we want to get an overview of the NFR classes that are documented in practice.

**RQ2** What NFR classes are documented in different application domains? Under this RQ, we analyze whether there is an observable difference between the application domains w.r.t. the documented NFR classes.

2. Nature of the NFR classes.

We analyze the NFR classes with respect to their nature (behavioral or representational) and their kind of behavior via three research questions:

**RQ3** How many NFRs describe system behavior? With this RQ, we want to better understand how many NFRs describe system behavior and how many describe the representation of a system (behavioral vs. representational) and whether this varies for different NFR classes.

**RQ4** Which system views do behavioral NFRs address? With this RQ, we want to better understand the relation between NFR classes and the system (modeling) views that the NFRs address, e.g., interface, architecture, or state behavior.

**RQ5** In which type of behavior theory are behavioral NFRs expressed? With this RQ, we want to better understand the relation between NFR classes and behavior theories used to express the NFRs, e.g., logical, timed, or probabilistic description.

3.2 Study Object

The study objects used to answer our research questions constitute 11 industrial specifications from 5 different companies for different application domains and of different sizes with 346 NFRs in total. We collected all those requirements that were explicitly labeled as “non-functional”, “quality”, or any specific quality attribute. The specifications further differ in the level of abstraction, detail, and completeness. We cannot give detailed information about the individual NFRs or the projects. Yet, in Table 1, we summarize our study objects, their application domain, and show exemplary (anonymized) NFRs as far as possible within the limits of existing non-disclosure agreements.

3.3 Data Collection and Analysis Procedures

To answer our research questions, we prepared the NFRs from our study object and then performed a classification and analysis. The procedure was performed by the first two authors in a pair. Both have over three years of experience in requirements engineering research and model-based development research.

3.3.1 Data Preparation

The NFRs from our study objects differ in their level of abstraction, detail, and completeness. Therefore, we went through the set of NFRs and processed each of them in either one of the following ways:

- **Full interpretation**: We considered the NFR as it is.
- **Sub interpretation**: We considered only a part of the NFR that we clearly identified as desired system property and disregarded the rest of the NFR (e.g., due to unnecessary/misleading information).
- **Split requirement**: We split the NFR into a set of singular NFRs because the original NFR addressed more than one desired property of a system.
- **Exclude from study**: We excluded the NFR if it was not in the scope of our study (e.g., process requirements), or if we were not able to understand the NFR due to missing or vague information.

In total, we excluded 56 requirements (≈ 16%) from the study and considered 76 requirements (≈ 22%) only partially. We split 97 requirements (≈ 28%) into an overall of 337 requirements. Together with the 117 requirements (≈ 34%) that we considered as they are, we ended up with a set of 530 requirements that we used for our classification.

3.3.2 Data Classification

We classified each of the 530 NFRs according to the following classification schemes:

- **Behavioral NFRs**: We included behavioral NFRs that express system behavior (e.g., timed, probabilistic).
- **Representational NFRs**: We included representational NFRs that describe system properties (e.g., interface, architecture).
- **Non-behavioral NFRs**: We included non-behavioral NFRs that do not express system behavior (e.g., availability).

In the data preparation phase, we split non-singular NFRs into singular NFRs. Thus, the final number of analyzed NFRs is 530.
Table 1: Overview of the study objects

<table>
<thead>
<tr>
<th>Spec.</th>
<th>Application Domain</th>
<th># ReqS</th>
<th># NFRs</th>
<th>% NFRs</th>
<th>Exemplary NFR (anonymized due to confidentiality)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>BIS (Finance)</td>
<td>200</td>
<td>61</td>
<td>30.5%</td>
<td>The availability shall not be less than [x]%%. That is the current value.</td>
</tr>
<tr>
<td>S2</td>
<td>BIS (Automotive)</td>
<td>177</td>
<td>40</td>
<td>22.6%</td>
<td>An online help function must be available.</td>
</tr>
<tr>
<td>S3</td>
<td>BIS (Finance)</td>
<td>107</td>
<td>5</td>
<td>4.7%</td>
<td>The maximal number of users that are at the same time active in the system is [x].</td>
</tr>
<tr>
<td>S4</td>
<td>ES/BIS (Travel Mgmt.)</td>
<td>38</td>
<td>14</td>
<td>36.8%</td>
<td>The [system] must run on [the operating system] OS/2.</td>
</tr>
<tr>
<td>S5</td>
<td>ES/BIS (Travel Mgmt.)</td>
<td>69</td>
<td>16</td>
<td>23.2%</td>
<td>It must be possible to completely restore a running configuration when the system crashes.</td>
</tr>
<tr>
<td>S6</td>
<td>ES (Railway)</td>
<td>35</td>
<td>14</td>
<td>40.0%</td>
<td>The delay between passing a [message] and decoding of the first loop message shall be ≤ [x] seconds.</td>
</tr>
<tr>
<td>S7</td>
<td>ES (Railway)</td>
<td>122</td>
<td>19</td>
<td>15.6%</td>
<td>The collection, interpretation, accuracy, and allocation of data relating to the railway network shall be undertaken to a quality level commensurate with the SIL [x] allocation to the [system] equipment.</td>
</tr>
<tr>
<td>S8</td>
<td>ES/BIS (Traffic Mgmt.)</td>
<td>554</td>
<td>128</td>
<td>23.1%</td>
<td>Critical software components shall be restarted upon failure when feasible.</td>
</tr>
<tr>
<td>S9</td>
<td>ES (Railway)</td>
<td>393</td>
<td>12</td>
<td>3.0%</td>
<td>The [system] will have a Mean Time Between Wrong Side Failure (MTBWSF) greater than [x] h respectively a THR less than [x]/h due to the use of [a specific] platform.</td>
</tr>
<tr>
<td>S10</td>
<td>ES (Railway)</td>
<td>122</td>
<td>31</td>
<td>25.4%</td>
<td>The [system] system shall handle a maximum of [x] trains per line.</td>
</tr>
<tr>
<td>S11</td>
<td>BIS (Facility Mgmt.)</td>
<td>24</td>
<td>6</td>
<td>25.0%</td>
<td>The architecture as well as the programming has to guarantee an easy and efficient maintainability.</td>
</tr>
</tbody>
</table>

Note: We distinguish BIS (Business Information Systems), ES (Embedded Systems), and hybrids of both. For reasons of simplicity, we group various domains (e.g. "Finance") according to single family of systems and use the term "application domain" for that classification.

**NFR class:** We used the quality model for external and internal quality of the ISO/IEC 9126 [16] to assign a quality characteristic to each NFR (Functionality-Suitability, Reliability-Maturity, . . . ; see [16] for details). In our study, the ISO/IEC 9126 quality characteristics represent the NFR classes we consider.

**System view:** We based our classification on Broy’s structured views [5, 6] to assign a system view to each NFR. As illustrated in Figure 1, structured views partition NFRs into representational NFRs that refer to the way a system is syntactically or technically represented, described, structured, implemented, or executed (e.g., NFR of S4, Table 1), and behavioral NFRs that describe behavioral properties of a system. Behavioral NFRs are further partitioned into NFRs that describe black-box behavior at the interface of a system (e.g., NFR of S10, Table 1) and NFRs that address a glass-box view onto a system describing its architecture (e.g., NFR of S8, Table 1), or its state behavior (e.g., NFR of S5, Table 1).

**Behavior theory:** Each behavioral NFR uses a certain behavior theory to express the desired properties of the system. We differentiate between the following classes of behavior theories for our classification:

- **Syntactic** The NFR is expressed by a syntactic structure on which behavior can be described (e.g., NFR of S2, Table 1).
- **Logical** The NFR is expressed by a set of interaction patterns (e.g., NFR of S8, Table 1).
- **Timed** The NFR is expressed by a set of interaction patterns with relation to time (e.g., NFR of S6, Table 1).

```
Σ 1 1
Σ 1.841 Σ 346 18.8%
```

4 The decision tree can be found under: http://www4.in.tum.de/~eckharjo/DecisionTree.pdf
During the pre-study, we also recognized multiple occurrences of NFRs following a common pattern. For example, many specifications contained an NFR following the pattern: "The system shall run/be installed on platform X". We identified a list of 13 of such patterns and assigned a common classification that we applied to all NFR instances following that pattern.\(^5\)

### 3.3.3 Data Analysis Procedures

To answer RQ1, we analyzed the distribution of NFRs with respect to the ISO/IEC 9126 quality characteristics. We provide two views onto this distribution. One detailed view that shows the distribution of NFRs with respect to all 27 quality characteristics contained in the standard and one coarse-grained view that shows the distribution of NFRs with respect to only 7 aggregated quality characteristics. In the aggregated quality characteristics, we subsumed low-level quality characteristics (such as Functionality-Suitability and Functionality-Accuracy) to their corresponding high-level quality class (Functionality in this case). We made one exception: we created for Functionality-Security an own class, as most other NFR classifications handle security separately. This results in the following aggregated list of quality characteristics: Functionality, Usability, Reliability, Security, Efficiency, Maintainability, and Portability (in the following we will refer to this list as ISO\(^a\) quality characteristics).

To answer RQ2, we analyzed the distribution of NFRs in the ISO\(^a\) quality characteristics with respect to the application domain of the corresponding system. To answer RQ3, we contrast the number of representational NFRs with the number of behavioral NFRs. To answer RQ4, we analyze the distribution of the behavioral NFRs with respect to interface, architecture, and state behavior. To answer RQ5, we analyze the distribution of the behavioral NFRs with respect to the behavior theory used to express them. For each RQ, we present the results for the set of all requirements and structured according to the ISO\(^a\) quality characteristics.

### 4. STUDY RESULTS

In the following, we report on the result for our research questions structured according to the research questions introduced in Section 3.

#### 4.1 Distribution of NFR Classes in Practice

**RQ1: NFR Classes**

Table 2 shows the number (count) and percentage of NFRs (relative to the total number of NFRs) for each quality characteristic. Table 3 further shows the distribution with respect to the ISO\(^a\) quality characteristics. As shown in Table 2, the two classes Functionality-Suitability and Functionality-Security stand out with in total 221 NFRs (\(\approx 41.7\%\)). Functionality-Suitability is defined as "the capability of the software product to provide an appropriate set of functions for specified tasks and user objectives" [16]. This essentially corresponds to a classical understanding of a functional requirement. Furthermore, we classified up to 40 NFRs (\(\approx 7.5\%\)) as Reliability-Maturity, Usability-Operability, or as Efficiency-Time Behaviour.

\(^5\)The complete list of patterns and the corresponding classification can be found under: http://www4.in.tum.de/~eckharjo/PatternList.pdf

<table>
<thead>
<tr>
<th>Quality characteristic</th>
<th>count</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality - Suitability</td>
<td>117</td>
<td>22.1%</td>
</tr>
<tr>
<td>Functionality - Security</td>
<td>104</td>
<td>19.6%</td>
</tr>
<tr>
<td>Reliability - Maturity</td>
<td>40</td>
<td>7.5%</td>
</tr>
<tr>
<td>Usability - Operability</td>
<td>40</td>
<td>7.5%</td>
</tr>
<tr>
<td>Efficiency - Time Behaviour</td>
<td>37</td>
<td>7.0%</td>
</tr>
<tr>
<td>Reliability - Reliability Compliance</td>
<td>29</td>
<td>5.5%</td>
</tr>
<tr>
<td>Efficiency - Resource Utilization</td>
<td>21</td>
<td>4.0%</td>
</tr>
<tr>
<td>Portability - Adaptability</td>
<td>21</td>
<td>4.0%</td>
</tr>
<tr>
<td>Portability - Installability</td>
<td>18</td>
<td>3.4%</td>
</tr>
<tr>
<td>Maintainability - Changeability</td>
<td>12</td>
<td>2.3%</td>
</tr>
<tr>
<td>Reliability - Recoverability</td>
<td>11</td>
<td>2.1%</td>
</tr>
<tr>
<td>Functionality - Functionality Compliance</td>
<td>10</td>
<td>1.9%</td>
</tr>
<tr>
<td>Usability - Learnability</td>
<td>10</td>
<td>1.9%</td>
</tr>
<tr>
<td>Functionality - Accuracy</td>
<td>9</td>
<td>1.7%</td>
</tr>
<tr>
<td>Usability - Usability Compliance</td>
<td>9</td>
<td>1.7%</td>
</tr>
<tr>
<td>Functionality - Interoperability</td>
<td>8</td>
<td>1.5%</td>
</tr>
<tr>
<td>Usability - Understandability</td>
<td>8</td>
<td>1.5%</td>
</tr>
<tr>
<td>Maintainability - Analyzability</td>
<td>7</td>
<td>1.3%</td>
</tr>
<tr>
<td>Reliability - Fault Tolerance</td>
<td>6</td>
<td>1.1%</td>
</tr>
<tr>
<td>Maintainability - Stability</td>
<td>4</td>
<td>0.8%</td>
</tr>
<tr>
<td>Portability - Replaceability</td>
<td>4</td>
<td>0.8%</td>
</tr>
<tr>
<td>Portability - Co-Existence</td>
<td>3</td>
<td>0.6%</td>
</tr>
<tr>
<td>Maintainability - Maintainability Compliance</td>
<td>1</td>
<td>0.2%</td>
</tr>
<tr>
<td>Usability - Attractiveness</td>
<td>1</td>
<td>0.2%</td>
</tr>
</tbody>
</table>

In the aggregated results shown in Table 3, one can see that the most common classification of NFRs is Functionality with around 27%. Furthermore, around 20% of all NFRs concern Security, 16% concern Reliability, and 13% concern Usability. Efficiency (\(\approx 11\%)\), Portability (\(\approx 9\%)\), and Maintainability (\(\approx 5\%)\) occur only to a small extent in our data.

**RQ2: Relation to Application Domain**

The results for RQ2 are given in Figure 2 showing the distribution of NFR quality characteristics with respect to the application domain of the corresponding system (Business Information System (BIS), Hybrid (ES/BIS), or Embedded System (ES)).

One can see a clear difference in the distribution of quality characteristics among the application domains. For example, for business information systems, we classified most NFRs as Security or Functionality, while for embedded systems, most NFRs are classified as Reliability. In hybrid systems, the distribution among the quality characteristics is more balanced compared with the other application domains.

Although we expected to see different distributions of NFR classes between application domains, we were surprised by the extent of this difference. We see these results as a strong argument for domain-specific handling of NFRs. In Section 6, we will discuss this in more detail.
4.2 Nature of NFR Classes

**RQ3: Amount of NFRs Describing System Behavior**

The results for RQ3 are shown in Figure 3. The table shows the distribution of behavioral and representational NFRs for all NFRs from our data set while the bar chart shows the distribution with respect to the ISO quality characteristics. More precisely, the bar chart shows the percentage of NFRs that we classified as black-box (black), glass-box (dark gray), or representational (light gray) within each ISO class.

More than half of each of the NFRs in the Functionality, Usability, Reliability, and Efficiency classes describe black-box behavior defined over the interface of the system. For example, most efficiency requirements describe desired or expected time intervals between events that are observable at the system interface. Reliability requirements often describe the observable reaction of the system at the interface if an error occurs within the system, such as “The [system] must have a mean time between failures greater than \([z]\) h”.

The only class where the largest share of NFRs is classified as glass-box behavior is Maintainability. That means, maintainability requirements, if they consider system properties, often describe the desired internal structure or behavior within this structure (glass-box), as for example the requirement “The configuration [of the system] shall be independent from the system software and application software”. However, a substantial amount of glass-box behavior can also be found in the Functionality, Reliability, Security, Efficiency, and Portability classes. Thus, NFRs within these classes also describe internal behavior, as for example the Portability requirement “The server software shall have the capability to run together with other applications on the same hardware whenever possible”.

Considering the amount of representational NFRs, one can see that the NFR classes Usability, Portability, and Security stand out. For usability and portability, this is as we expected. Usability requirements often describe representational aspects of the user interface with the goal to support the user in understanding and controlling a system, as for example the requirement “[The] GUI shall provide a common look and feel whenever possible”. Portability requirements demand the system to be represented in a way that it fits a specified environment, as for example the requirement “The system shall run on platform X”. However, for security, we did not expect such a high portion of representational NFRs. Therefore, we analyzed these in detail and found that many representational NFRs in the Security class contain a reference to a standard. For example, we found a high number of NFRs stating, “The security class of the interface to system X with respect to data confidentiality is high”. Excluding those NFRs that reference standards from the results, around 54% of security NFRs describe black-box behavior, 39% describe glass-box behavior, and only 7% describe representational aspects. This shows that some aspects of security are visible at the interface, as for example user authentication, and some aspects are internal to the system, as for example an encrypted communication within sub-systems.

Another point interesting to us was that none of the NFR classes is exclusively black-box, glass-box, or representational. For example, in the Functionality class, most NFRs describe black-box behavior. However, around 31% of the NFRs describe glass-box behavior and 17% describe representational aspects. This is because the Functionality class does not only include behavior over the interface, but also internal behavior like “A system component shall save a user’s edits whenever possible”, and also representational aspects like “The backup data must be stored according to [the company’s] policies.”
RQ4: Distribution of Behavioral NFRs w.r.t. System Views

The results for RQ4 are shown in Figure 4. The table shows the distribution of NFRs with respect to the system view they address while the bar chart shows this distribution with respect to the ISO quality characteristics. More precisely, the bar chart shows the percentage of NFRs that we classified as interface (black), architecture (dark gray), or state (light gray). For RQ4, we considered only behavioral NFRs and neglected NFRs classified as representational, as they do not describe behavior.

Quantitative results of RQ4:
68.9% of all behavioral NFRs describe behavior over the interface of the system, 21.5% describe architectural behavior, and 9.6% describe behavior related to states of the system.

We can see that in the Functionality, Usability, Reliability, Security, and Efficiency classes most behavioral NFRs are classified as interface. For the Maintainability and Portability classes, the most common classification is architecture. Usability is the only NFR class without any NFR classified as architecture. We can further see that all NFR classes but Efficiency contain NFRs that describe state-related aspects, as for example the Functionality requirement “[The system] must ensure that submitted offers can neither be modified nor deleted”. This shows that behavioral NFRs describe externally visible behavior but also behavior concerning the architecture (see structuring the functionality by functions [4]) or state-related behavior (see operational states of a system [27]). For example, in the Security class, there are NFRs that describe behavior over the interface like “There has to be an authentication mechanism”, some NFRs describe architectural behavior like “[The system] must provide intrusion detection mechanisms”, and some describe state-related aspects like “The password shall be valid for at most 30 days”.

RQ5: Distribution of Behavioral NFRs w.r.t. Behavior Theories

The results for RQ5 are shown in Figure 5. The table shows the distribution of NFRs with respect to the behavior theory they use and the bar chart shows this distribution with respect to the ISO quality characteristics. More precisely, the figure shows the percentage of NFRs that we classified as syntactic, logical, timed, probabilistic, or probabilistic and timed (from black to white). For RQ5, we considered only behavioral NFRs and neglected NFRs classified as representational as they do not describe behavior.

Quantitative results of RQ5:
Most behavioral NFRs are logical (69.9%), 18.2% are timed and/or probabilistic, and only 11.9% are syntactic.

Over all NFR classes, most NFRs are logical (around 69.9%), while 13.6% are timed, 11.9% are syntactic, 2.8% are probabilistic and timed, and 1.8% are probabilistic. Most timed and also probabilistic and timed NFRs belong to the Efficiency class. Moreover, the Reliability class stands out, as it also contains many timed, probabilistic, and timed and probabilistic NFRs.

4.3 Summary of Results

Figure 6 provides a consolidated quantified view on our overall results.
In conclusion, most NFRs address interface behavior, mostly expressed by logical or timed assertions. The NFR classes Usability, Security, and Portability include, in contrast to the other classes, a high portion of representational NFRs. Furthermore, all classes but Usability contain architectural aspects (see column Architecture), while the highest percentage of those NFRs are in the Maintainability class.

5. THREATS TO VALIDITY

In the following, we discuss the threats to validity and mitigation measures we took. We discuss them along the different problems as they arose during our work.

5.1 Data Representativeness Problem

Inherent to the nature of our study is the data representativeness on which we built our analysis. The concerns range from the representativeness of the way the NFRs are specified to the completeness of the data as it currently covers only the particularities of selected industrial contexts. We cannot mitigate this threat but consider our data set large enough to allow us to draw first conclusions on the state of the practice. The relation to existing evidence (see Section 7.1) additionally strengthens our conclusions.

5.2 NFR Selection Problem

We collected only those requirements explicitly labeled as non-functional or quality. With this selection procedure, some relevant NFRs might have been missed or irrelevant ones might have been included. To address this problem, we plan to perform the classification on the whole data set as future work, including functional and non-functional requirements.

5.3 Preparation Problem

In our data preparation phase, we excluded NFRs from the study if they were not in the scope of our study (e.g., process requirements) or if we were not able to understand them (due to missing or vague information). This exclusion process could threaten the overall conclusion validity, but as we excluded only about 16%, we do not consider this as a major threat.

5.4 Classification Problem

Prior to our study, we performed a pre-study with several independent classification rounds [12]. The inter-rater agreement between the independent raters was, however, so low that we had to conclude that the classification dimensions are not clear enough. To resolve this issue, we performed several refinements of the classification and created a decision tree and a pattern catalogue that supports the classification process [12]. In the end, we did the classification in a pair of researchers and individually discussed each NFR.

5.5 Representation Problem

Although classifying in a pair of researchers, we still faced the representation problem discussed by Glinz [14], which threatens the internal validity. If an NFR stated “The system shall authenticate the user”, we classified it as black-box interface, and logical as it describes a black-box behavior over the interface. However, if an NFR stated “The system shall contain an authentication component”, we classified it as glass-box architecture and logical as it requires an external component for authentication.
5.6 Contextualization Problem

We consider the reliability of our conclusions to be very much dependent on the possibility to reproduce the results, which in turn is dependent on the clearness of the context information. The latter, however, is strongly limited by NDAs that too often prevent providing full disclosure of the contexts and even the project characteristics. To mitigate this threat, we anonymized the data as much as possible and disclosed all information possible within the limits of our non-disclosure agreements.

6. DISCUSSION

Based on the results, we identified a set of insights which we discuss in the following paragraphs.

NFRs are not non-functional. It is commonly acknowledged that functional requirements describe logical behavior over the interface of the system. From a broader view, one could even say that functional requirements describe any kind of behavior over the interface of the system, including timing and/or probabilistic behavior. From this perspective, we conclude that many of those NFRs that address system properties describe the same type of behavior as functional requirements do (see column Interface in Figure 6). This is true for almost all NFR classes we analyzed; even for NFR classes which are sometimes called internal quality attributes (e.g., portability or maintainability) [20]. Hence, we argue that—at least based on our data—most “non-functional” requirements describe functional aspects of a system and are, thus, basically not non-functional. From a practical point of view, this means that most NFRs can be elicited, specified, and analyzed like functional requirements. For example, NFRs classified as black-box interface requirements, are candidates for system tests. In our data set, system test cases could have been specified for almost 51.5% of the NFRs.

Functional requirements are often labeled as NFRs. Moreover, functional requirements in the classical understanding were often labeled as NFRs in our examined specifications. We classified 22.1% of our overall NFR population as Functionality-Suitability, which is a quality characteristic that addresses the functionality of a system (“The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives” [16]). Given that NFRs are usually not tested and analyzed as thoroughly as functional requirements [1, 3, 26], this means that, one out of five NFRs in our data set elude a thorough analysis process just because they are labeled as NFRs.

NFRs are often specified by reference to standards. As already indicated within our results for RQ3, we realized that several examined NFRs describe requirements by pointing to a standard (e.g., company style guides or safety standards). More specifically, 68 of 530 NFRs (∼13%) contained references to standards. We classified these as representational since we were not able to access these standards due to availability and time constraints. However, these standard-referencing NFRs might be interesting to explore in future investigations. On the one hand, they allow a concise specification; on the other hand, they introduce much implicitly necessary knowledge and assume that the reader of the specification has knowledge about and access to those standards.

Only few NFRs deal with architectural aspects. While in literature the relation of NFRs to architecture and architectural constraints of a system is often emphasized [10, 28, 30], the NFRs of our sample dealt with architecture only to a small degree (see column Architecture in Figure 6). Only for Efficiency, Maintainability, and Portability, roughly one quarter of the NFRs considered architectural aspects of a system. Following this, we argue that—at least based on our data—only few NFRs actually describe architectural aspects of a system. It is an interesting point for future research to mirror our findings with the notion of architecturally significant requirements (ASRs) [7]. ASRs are those requirements which have a measurable impact on a software system’s architecture. They are often difficult to define and articulate, tend to be expressed vaguely, are often initially neglected, tend to be hidden within other requirements, and are subjective, variable, and situational [7]. Certainly, all NFRs that we classified as addressing the system view architecture can be considered as ASRs. However, also NFRs that we classified as addressing the system view interface or state may have an impact on the architecture, as for example the requirement “the system should provide five nines (99.999 percent) availability”. The difference is that NFRs addressing the system view architecture make the impact on the architecture explicit. For other NFRs, an architect needs to decide whether they are ASRs or not.

No NFR class is uniquely affiliated with only one behavior characteristic. Our analysis shows that none of the considered NFR classes is characterized by only one specific system view or behavior theory. Accordingly, most NFR classes contain representational and behavioral NFRs which address all system views and using all behavior theories. While a classification of NFRs according to quality characteristics may be helpful to express the intent of an NFR, the quality characteristic should not determine how an NFR is specified, implemented, or tested. This decision should rather be made based on the addressed system view and the behavior theory used to express the NFR.

The application domain influences NFRs. As our results indicate, the application domain of the corresponding system influences the relevance of NFR classes. We therefore conclude that specification and analysis procedures should be customized for different application domains. For example, in the embedded systems domain, the need for probabilistic analysis techniques is stronger compared with the business information systems domain due to the larger amount of reliability NFRs that are often described in a probabilistic manner. On the other hand, for business information systems, we should support specification techniques that integrate functional requirements and behavioral NFRs, since around 70% of the NFRs from BISs were classified as functionality or security (excluding the NFRs referencing standards from the security class, most NFRs in the security class concern the interface), which describe logical interface behavior to a large extent.
NFRs are specified at different levels of abstraction. In our data set, we found NFRs at different levels of abstraction varying in their degree of detail and completeness. NFRs ranged from high-level goal descriptions like “The availability shall not be less than [5]%” to very concrete and detailed descriptions of behavior like “The delay between passing a [message] and decoding the first loop message shall be ≤ [5] seconds”. This is in tune with the view of Pohl [23]; He states that non-functional requirements are underspecified functional requirements. In a development process, high-level NFRs need to be refined to detailed functional requirements. To make this refinement explicit, we need an approach for relating high-level NFRs (or quality goals) to low-level functional requirements. A first approach in this direction is proposed by Broy in his recent work [6].

7. CONCLUSIONS
In this paper, we reported on a study where we analyzed and classified 530 NFRs extracted from 11 industrial requirements specifications with respect to their kind and their relation to system views and behavior theories. Our goal was to gain a better understanding on the nature of system-specific NFRs, i.e., those NFRs that address system properties. We were able to show, for example, that most of the NFRs in our sample actually describe black-box interface behavior. Our overall conclusion is that NFRs are in their nature not non-functional. Therefore, we argue that many so-called NFRs can be handled similarly to functional requirements and, thus, can be integrated into a seamless software engineering development process.

7.1 Relation to Existing Evidence
Our results show various relations to existing evidence. For instance, during our classification, we faced all three problems described by Glinz [14]. We also experienced same or similar terminological confusions on NFRs as reported by Ameller et al. [1]. In particular, we found that categories such as availability were often misinterpreted in the documents and used in different ways, e.g., as performance. Furthermore, they found that the four NFR classes most important to software architects were performance, usability, security, and availability (in that order). We could support their results via quantitative results: Their four NFR classes are in our list of the top four NFR classes (in a different order). Finally, we also resemble the results of Mairiza et al. [19] with respect to the five most frequently mentioned NFR classes in literature.

Apart from supporting existing evidence, we provide first empirical evidence on what non-functional (system) requirements are in their nature by analyzing and classifying them with respect to various facets. Summarizing our findings, we conclude that most so-called “non-functional” requirements in our sample describe functional aspects of a system and are, thus, essentially not non-functional.

7.2 Impact/Implications
Our results strengthen our confidence that many requirements that are currently classified as NFRs in practice can be handled equally to functional requirements, which has both a strong theoretical and practical impact. Existing literature (e.g., [1, 3, 9, 26]) indicates that the development process for a requirement differs depending on whether it is classified as “non-functional” or “functional”. In contrast to functional requirements, requirements classified as NFR are often neglected and properties like testability are not supported. In industrial collaborations, we have also seen that NFRs and functional requirements were documented in separate documents, which has led to failing acceptance tests performed by an external company. Our results suggest that this separation is artificial to a large extent. We argue that treating NFRs the same as functional requirements would have major consequences for the software engineering process. However, there are currently no empirical studies that investigate this argument in detail.

A long-term vision that emerges from our results is that we might be able to better integrate NFRs into a holistic software development process in the future. Such an integration would yield, for instance, seamless modeling of all properties associated with a system—no matter if they are functional or non-functional in a classical understanding. The benefits of such an integration include that NFRs would not be neglected during development activities, as it is too often current state of practice; from an improvement in the traceability of requirements over an improvement of possibilities for progress control to an improvement of validation and verification.

7.3 Future Work
Our analysis is based on an inherently incomplete set of requirements specifications gathered from practical environments. Hence, our study can be considered as a first attempt to improve the understanding on the nature of NFRs from a practical perspective. This has certain implications on the validity of our results (see Section 5). However, they still provide a suitable basis to draw first conclusions, which need to be strengthened via additional studies; for instance, by increasing the sample size, by taking into account further application domains, but also by including functional requirements into the analysis. So far, our results are suitable to trigger critical, yet important discussions within the community.

We are planning three concrete next steps based on our data set: First, we will include the remaining 1495 functional requirements (the ones not labeled as “non-functional” or quality attribute) in our study. Second, we are planning to advance the integration of NFRs into software development by providing specification blueprints (based on an integrated model) for practitioners. Third, as discussed in Section 7.2, we will investigate the consequences of labeling a requirement as “NFR” for the development process. We expect to find consequences for how requirements labeled as NFRs are tested or when they are considered in the development process.

Acknowledgements
We would like to thank M. Broy, E. Jürgens, M. Junker, B. Penzenstadler, and S. Smith-Eckhardt for their helpful comments on earlier versions of this work. Furthermore, we thank the anonymous reviewers for their detailed feedback on our work.

8. REFERENCES
20th IEEE International Requirements Engineering Conference (RE), 2012.


