A Metamodel for Tracing Non-Functional Requirements

M. Kassab¹, O. Ormandjieva², M. Daneva³
¹,² Concordia University, Canada, {moh_kass, ormandj}@cse.concordia.ca
³ University of Twente, The Netherlands, m.daneva@utwente.nl

Abstract

The tendency for Non-Functional Requirements (NFRs) to have a wide-ranging impact on a software system, and the strong interdependencies and tradeoffs that exist between NFRs and the software project, leave typical existing software modeling methods incapable of integrating them into the software engineering. In this paper, we propose a metamodel which explicitly models FRs, NFRs, their refinements and their interdependencies. The metamodel, which is independent from any programming paradigm, is further transformed into a relational model which facilitates NFRs traceability using tracing queries implemented through “Datalog” expressions. The approach is illustrated on a case study.

1. Introduction

In the early phases of software development, user requirements are established based on an analysis of business goals and the application domain. Subsequently, architectures of the desired systems are designed and implemented. During this development process, requirements are usually exposed to many changes as the availability of knowledge on the system being developed increases [10]. Traceability, defined as “the ability to describe and follow the life of a requirement in both a forwards and backwards direction” from inception throughout the entire system’s life cycle, provides useful support mechanisms for managing requirement changes during the ongoing change process [24, 25]. Moreover, the extent to which traceability is exploited is viewed as an indicator of system quality and process maturity, and is mandated by many standards [23].

In practice, many organizations either focus their traceability efforts on functional requirements (FRs) [21] or else fail entirely to implement an effective traceability process [14, 16]. Tracing NFRs has, by and large, been neglected. This is mainly because NFRs tend to become scattered among multiple modules when they are mapped from the one-dimensional requirements domain to the n-dimensional solution space. Furthermore, they can often interact, in the sense that attempts to achieve one NFR can help or hinder the achievement of other NFRs at a certain of functionality. Such an interaction creates an extensive network of interdependencies and trade-offs between NFRs which is not easy to trace [9]. Nevertheless, reports consistently indicate that neglecting NFRs can lead to catastrophic project failures, or at the very least to considerable delays and, consequently, to significant increases in the final cost. Valid examples are: London Ambulance System (LAS) in 1992 [2], Mars Climate Orbiter in 1998 [18], Therac 25: The Medical Linear accelerator [20] and the Mercedes A-Class (1997) [22].

In this paper, we propose a metamodel which explicitly captures the concepts of NFRs, FRs and their relations throughout the software development process, and which is independent of any programming paradigm. The metamodel can be enhanced to provide additional properties and concepts, and instantiated to define a customized traceability model with respect to the required programming type (e.g. object-orientation, procedural programming, etc.).

We transform the proposed metamodel into a relational model [28] based on predicate logic and set theory. We then use Datalog expressions to implement queries to represent tracing information. Query evaluation with Datalog is sound and complete, and can be performed efficiently, even for large databases. In addition, Datalog supports Recursive Closure Operations which make it possible to trace through multiple levels of refinements within the software development process. The discussions in this paper are illustrated through examples from an Email Application case study.

The remainder of this paper is organized as follows: Section 2 provides a brief overview of related work. Section 3 introduces the requirements metamodel. Section 4 presents the relational model and implementation of tracing queries. Section 5 proposes a traceability model using the metamodel.
and the relational model. Section 6 concludes the paper with an early analysis of the applicability and limitations of the approach and some suggestions for future work.

2. Related work

Although prior work on tracing NFRs has been rather limited, a number of traceability approaches have in fact been developed to support related activities while incorporating NFRs in software engineering processes.

In [19], the authors adopt the NFR Framework [8] to show how a historical record of the treatment of NFRs during the development process can also serve to systematically support evolution of the software system. The authors treat changes in terms of (i) adding or modifying NFRs, or changing their relative importance, and (ii) changing design decisions or design rationale. While this study has provided some support for extensions to the NFR Framework, particularly in representing changes to goal achievement strengths, the impact of changes to functional models on non-functional models, and vice-versa, has yet to be discussed.

In [16, 17], the authors propose an approach named Goal Centric Traceability, a holistic traceability environment which provides systems analysts with the means to manage the impact of functional change on NFRs. Nevertheless, the impact of changes to an NFR on other NFRs and the functional model is not solved with this solution.

In [5, 11, 15], early NFR integration is accomplished by extending UML models to integrate NFRs into functional behavior. Although consideration of the compositional process at the meta level is essential for intra-phase traceability, these approaches only model certain NFRs (e.g. response time, security) in a way that is not necessarily applicable to other requirements. Indeed, there is no single, formal method available that is well suited to defining and analyzing numerous system NFRs.

Many other initial approaches have been introduced by researchers active in the Requirements Engineering, Product Line Engineering, and Aspect-oriented Software Engineering communities to address the traceability of NFRs [1, 3, 4, 6, 7, 10, 12, 13, 17, 23, 25, 26, 29]. These approaches have three important limitations. First, tracing is either tackled within a phase or it does not cover the entire life cycle. Second, the traceability model that is applied is usually focused on specific programming paradigm elements. Third, these approaches use coarse-grained entities for tracing purposes, which that there is a risk of imprecise change impact analysis, which in turn results in imprecise estimates of the cost and time involved in implementing a requirement change. The specific challenges faced in state-of-the-art traceability practice are described in more detail in [23].

This paper offers a solution to the open research problems discussed in this section. The proposed here metamodel is well suited for defining and analyzing numerous NFRs, the impact of changes in a NFR upon other NFRs, NFRs impact on the FRs and vice-versa traceable over the entire life cycle.

3. Explicit NFR modeling

Obviously, NFRs can be included in the project in various phases of the life cycle, and traces should be supported within and across life-cycle phases. In order to explicitly reason about the traceability of NFRs, it is necessary that the corresponding NFRs and their relations be explicitly modeled. In this section, we present the requirement relations metamodel, which is schematically represented in Figure 1. Therein, NFRs are modeled as parts of a requirements group which is a part of a requirements model.

The left-hand side of Figure 1 presents the functional models, and shows that an FR is realized through the various phases of development by many functional models (e.g. in the object-oriented field, a use-case model is used in the requirements engineering phase, an analysis model is used in the requirements analysis phase, a design model is used in the software design phase, etc.). Each model is an aggregation of one or more artifacts (e.g. a use-case diagram and a use case for the use-case model, a domain model diagram and a system sequence diagram for the analysis model, a class diagram and a communication diagram for the design model). The artifact by itself is an aggregation of elements (e.g. a class, an association, an inheritance, etc. for the class diagram). Modeling artifacts and their elements in this way gives us the option of decoupling the task of tracing NFRs from a specific development practice or paradigm.

The righthand side of Figure 1 shows the part of our metamodel that is used to model NFRs and their relations. Four relations are identified, namely, association, decomposition, operationalization and interactivity.

**Association.** NFRs do not represent stand-alone goals, as their existence is always associated with other goals or concepts. In this work, we define three
association points with which an NFR and its derived solutions (the so-called operationalizations) can be associated throughout the software development process:

- The FR (and any element belonging to an artifact modeling functionality in any phase): This refers to the context for functionality-related NFRs.
- Resource: This refers to external-entity-related NFRs such as scalability.
- Project: This refers to NFRs which provide a precise context to the project or development process.

**Decomposition.** This refers to the relation that decomposes a high-level NFR into more specific sub-NFRs. *In each decomposition*, the offspring NFRs can contribute partially or fully towards satisfying the parent. The decomposition can be “ANed” (all NFR offspring are required to achieve the parent NFR goal) or “ORed” (it is sufficient that one of the offspring be achieved instead, the choice of offspring being guided by the stakeholders). In the case of “ANed”, as in the security example (see Figure 4), all the sub-NFRs are also associated with the FR with which the parent NFR is associated. For example, the set of association points with security is a subset of the set of confidentiality, integrity, or availability association points. In the case of “ORed”, then only the sub-NFRs that are selected by stakeholders will be associated with the FRs with which the parent NFR is associated. Figures (2b) and (2c) illustrate the two situations. The question mark notation “?” in (2c) indicates that a further contribution from the stakeholders is required to determine the existence of the relation.

**Operationalization.** This refers to the relation that refines the NFR into solutions in the target system that will satisfice the NFR. These solutions provide operations, processes, data representations, structuring, constraints, and agents in the target system to meet the needs stated in the NFRs. Similar to decomposition, operationalization can be ANed or ORed. In the confidentiality example (see Figure 4), either implementing authorization or the use of additional ID is sufficient, in which case both operationalizations are ORed. We note, however, that the existence of an association between a parent NFR and a functionality (e.g. security and read an email message) does not necessarily imply that an association exists between those operationalizations which is derived from the parent NFR and artifact or part of it, which in turn is derived from the FR (e.g. the use of additional ID and a method implementing the read an email message functionality). System analysts have, then, to check whether or not this existence dependency is valid. Figure (2a) illustrates this situation.

![Figure 1: Metamodel for NFRs, FRs, and their relations.](Image)
Interactivity NFRs by themselves do not interact, as they represent static goals to be achieved. However, their associations with functionalities could interact, in that attempts to achieve one NFR at a certain association point can hinder (negative interaction) or help (positive interaction) the achievement of other NFRs at the same association point, e.g. security and performance at read an email message functionality (see Figure 4). Two NFRs negatively affect one another if they can be traced to the same association point and, at the same time, compete for the same resources. This can be illustrated with the security and performance example above (the resource is CPU time).

Positive interaction would involve an offspring NFR and its parent NFR in the case of ANEd decomposition. In ORed decomposition, only the sub-NFRs, which are selected by the stakeholders, will positively affect the parent NFR. The interaction is not necessarily a symmetrical relation.

If one association $A_1$ (between security and read an email message, for example) has an interaction with another association $A_2$ (performance and read an email message) (see Figure 4), then at least one sub-NFR refined from NFR$_1$ (confidentiality refined from security) has an association (confidentiality and read an email message) that interacts with $A_2$. This situation is generalized in Figure (2d).

4. Relational data model for tracing requirements

While the metamodel described in section 3 is a useful way to understand the design structure of the concepts, it is not considered a suitable basis for retrieving data on the objects that are instantiated from this model. Thus, the model has to be transformed into another model which facilitates querying on the information. The relational model is extremely useful as a mapping vehicle because it is based on a single data modeling concept, namely the relation.

For the purposes of this work, we decided to use Datalog expressions [28] to operate on one or more relations to yield another relation which would present the desired results. Datalog (a subset of Prolog) is a language of facts and rules, as well as a logic-based query language for the relational model. Query evaluation with Datalog is sound and complete. Figure 3 presents the schemas for the relations corresponding to the subset of concepts shown in Figure 1. The relations are intended to hold information collected by stakeholders at different stages of the development cycle.

The discussion in this section will be illustrated through examples application system. The system requires its users to have a certain level of privileges to read an email message. The privileges are granted automatically upon successful authentication. In this work, we will limit the scope of the discussion to two functionalities, namely, read an email message and send email. Figure 4 (see Appendix 1) presents these two main pieces of functionality decomposed into elements of use-cases, events and methods. Three NFRs are also presented: security, performance and scalability.

While populating the relations, it is hard to ensure the completeness of the information as the majority of the instances of the relations are not directly stated by stakeholders but they hold as valid relations by induction. For example, security could be known as...
being associated with read an email message functionality. Confidentiality which is derived from “ANed” decomposition from security is also associated with read an email message according to figure (2b). This information on confidentiality association could be missed when populating the NFR_ASSOCIATION relation yet this relation has to be traced upon possible related requested changes in requirements. Our tracing mechanism considers this situation and is implemented so that it provides the suitable solution.

We identify four critical areas in which NFRs require traceability support. These areas are discussed in the following sub-sections.

[Figures and tables are not displayed.]
In the email system (see Figure 4), if a change is requested to a security requirement, then the above query expression will retrieve the read an email message functionality, all derived main and alternative scenarios, and the events select a message and open the selected message, as well as the methods m3 and m4.

4.3 Impact of changes to NFRs on lower/higher-level NFRs

The change to one NFR can migrate down to offspring NFRs or up to parent NFRs in a recursive manner through the decomposition links. This type of traceability enables the analyst to understand the impact of lower-level change on high-level goals, and vice versa. The following Datalog [28] expression implements this query:

\[
\text{TEMP}_1 (B, C) \leftarrow \text{NFR}\_\text{DECOMPOSITION} (A, B, C, D), B = \text{“NFR\_CHANGED”} \\
\text{TEMP}_1 (B, C) \leftarrow \text{NFR}\_\text{DECOMPOSITION} (A, B, C, D), C = \text{“NFR\_CHANGED”} \\
\text{TEMP}_1 (B, C) \leftarrow \text{NFR}\_\text{DECOMPOSITION} (A, B, C, D), \text{TEMP}_1 (X, B) \\
\text{RESULT}(X) = \text{TEMP}_1(X, Y), X = \text{“NFR\_CHANGED”} \\
\text{RESULT}(Y) = \text{TEMP}_1(X,Y), Y = \text{“NFR\_CHANGED”}
\]

In the email system (see Figure 4), if a change is requested to a space requirement, then the above query expression will retrieve the primary space, secondary space, and performance requirements.

4.4 Impact of changes on interacting associations

To complete intra-model traceability, it is necessary to establish traces between interacting NFRs at certain association points (interacting associations). The following Datalog expression implements this query:

\[
\text{RESULT}(Y) \leftarrow \text{NFR}\_\text{INTERACTION} (X,Y,Z,W), Z = \text{“CHANGED\_NFR”} \\
\text{RESULT}(Z) \leftarrow \text{NFR}\_\text{INTERACTION} (X,Y,Z,W), Y = \text{“CHANGED\_NFR”}.
\]

In the email system (see Figure 4), if a change is requested to a space requirement at read email message functionality, then the above query expression will retrieve the security requirement at that functionality.

5. Traceability Model

NFR tracing occurs through three distinct activities: requirement development, impact detection, and evaluation/decision-making. Each phase ensures that FR and NFRs are treated jointly and in an integrated fashion. These activities are depicted in Figure 5.

Impact detection is dependent on the effectiveness of the traceability mechanism in establishing correct links between functional and non-functional models and within their corresponding hierarchical models. Triggered by a change request, the potentially impacted area has to be identified, and then the corresponding query should be executed. Once the retrieval algorithm has returned a set of potentially impacted requirements/elements, the evaluation phase can commence. To analysts, this means they can now filter the retrieved requirements/elements to remove any non-relevant ones. A decision on any accepted change in any of the retrieved data should be recorded in the corresponding relations.

It is important to note that one change request can establish a chain of other requests. For example, the need to change one FR may generate the need to accept changes to other NFRs. In response to the NFR changes, the analysts may well see a need to change further sub-NFRs or interacting NFRs. We are currently working on establishing a formal algorithm for the change process.

![Figure 5. NFR-tracing activities](image-url)
6. Conclusion

In this paper, we propose a metamodel for requirement relations, and we propose a traceability mechanism under the umbrella of a relational model to track the allocation of requirements to system components, and control changes to the system. One of the advantages of our approach is that it forces system analysts to think about and capture the hierarchical relations within NFRs, the hierarchical relations within FRs, and the relations between NFR and FR hierarchies. Our approach helps systems analysts understand the relationships that exist within and across NFRs in the various phases of development. The paper proposes a method for tracing a change applied to an NFR in the traceability model, which results in a “slice” of the model containing all model entities, and makes them immediately reachable from that NFR within the hierarchy.

As this evaluation is an early one, our approach has not been validated using an empirical research method. We plan to remedy this by carrying out case studies. At this time, we have provided a “proof-of-concept” (as it is called in [27]). The email system case study used for illustration purposes in this paper is only representative in terms of its size and level of complexity for a small information systems project. It remains to be seen to what extent our experience with our approach as applied to this application can be transferred to other project settings, for example business information systems or large embedded system projects.

References


Appendix 1

Figure 4. Illustration of FR and NFR relations through the Email System