Relational-Model Based Change Management for Non-Functional Requirements: Approach and Experiment

M. Kassab, O. Ormandjieva
Department of Software Engineering and Computer Science
Concordia University
Montreal, Canada
{moh_kass , ormandj}@cs.concordia.ca

M. Daneva
Department of Information Systems
University of Twente
Enschede, the Netherlands
m.daneva@utwente.nl

Abstract – In software industry, many organizations either focus their traceability efforts on Functional Requirements (FRs) or else fail entirely to implement an effective traceability process. Non-Functional Requirements (NFRs) such as security, safety, performance, and reliability are treated in a rather ad hoc fashion and are rarely traced. This is mainly because of the unique nature of NFRs. They are subjective, relative and they tend to become scattered among multiple modules when they are mapped from the requirements domain to the solution space. Furthermore, NFRs can often interact, in the sense that attempts to achieve one NFR can help or hinder the achievement of other NFRs at particular software functionality. Such an interaction creates an extensive network of interdependencies and tradeoffs among NFRs which is not easy to trace. In a previous work, we proposed a conceptualization of NFRs through the NFRs Ontology. In this paper, we extend the previous work by proposing a change management mechanism for tracing the impact of NFRs on the other constructs in the ontology such as FR or NFR operationalization and vice versa, and providing a traceability mechanism using Datalog expressions to implement queries on a relational model-based representation for the ontology. The proposed traceability queries are then evaluated through a multi-project variation quasi-experiment on regression testing conducted in the industry.

I. INTRODUCTION

Software systems are characterized both by their functional behavior (what the system does) and by their nonfunctional behavior (how the system behaves with respect to some observable attributes like reliability, reusability, maintainability). In the software market place, in which functionally-equivalent products compete for the same customer, Non Functional Requirements (NFRs) become more important in distinguishing between the competing products. However, in practice, NFRs receive little attention relative to Functional Requirements (FRs) [1]. This is mainly because of the nature of these requirements which poses a challenge when taking the choice of treating them at an early stage of the development process. NFRs are subjective, relative and they tend to become scattered among multiple modules when they are mapped from the requirements domain to the solution space. Furthermore, NFRs can often interact, in the sense that attempts to achieve one NFR can help or hinder the achievement of other NFRs at particular software functionality. Such an interaction creates an extensive network of interdependencies and tradeoffs among NFRs which is not easy to trace or estimate [2].

In a previously published work [3], we proposed a formal model for NFRs and their relations. The model was captured though a Common Foundation for NFRs which was realized by developing the NFRs Ontology. A knowledge-based representation such as the one we presented in [3], is necessary to support the traceability of NFRs within a system and to provide practitioners and researchers with a valuable alternative to current requirements engineering techniques.

The research presented in this paper reports on our first usage of the NFRs Ontology as a vehicle towards supporting those requirements engineering (RE) activities that pertain to NFRs. In particular, the purpose of this work is to propose a mechanism to improve the NFRs traceability practice. In software industry, many organizations either focus their traceability efforts on FRs [1] or else fail entirely to implement an effective traceability process [4]. NFRs such as security, safety, performance, and reliability are treated in a rather ad hoc fashion and are rarely traced. Furthermore, the tendency for NFRs to have a global impact upon the software system necessitates the need to create and maintain an overwhelming number of traceability links. On the other hand, the appropriate support for NFRs traceability proposed in this paper can return significant benefits to an organization through helping analysts understand the impact of a proposed change upon critical system qualities and enabling them to maintain these qualities throughout the lifetime of a software system.

The research questions we address in this work are: What are the critical areas requiring traceability attention when dealing with change management of NFRs? How are these areas mapped to the concepts and relationships defined in the NFRs Ontology?

In this paper, we present a formal implementation of the answers derived from the above questions. The formal implementation was realized through Datalog queries [5] on a relational model-based representation for the NFRs ontology [3]. The proposed traceability for NFRs was further evaluated through a multi-project variation quasi-experiment on improving the quality of the regression testing conducted at NOKIA office in Montreal.

The remainder of this paper is organized as follows: Section II provides a brief overview of related work. Section III summarizes the NFRs Ontology, Section IV presents the

relational model and implementation of tracing queries using Datalog expressions. Section V provides a discussion and evaluation of applicability of the proposed traceability approach to improving the effectiveness and efficiency of regression testing. Section VI concludes the paper and outlines the directions of the future work.

II. 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 [6], the authors adopt the NFR Framework [2] 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 [4] and [7], 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.

Many other initial approaches have been introduced by researchers active in the RE, product line engineering, and aspect oriented software engineering communities to address the traceability of NFRs [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19] and [20]. In our review of these approaches, we observed that they have three important limitations. First, tracing is tackled within a specific phase or phases, and 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 is risky from the point of view of the precision of 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 [17].

This paper implements and evaluates a solution to the limitations discussed in this section. The solution rests on the NFRs Ontology proposed in [3] that 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. The NFRs Ontology is presented in the next section.

III. NFRS ONTOLOGY

The NFRs Ontology [3] defines (shared) meaning of a set of concepts for the NFRs domain. This can be used to improve communication and interaction among people, or even among systems. The ontology has an important core about NFRs model, but also addresses areas such as requirements and software architectures.

The NFRs Ontology contains many concepts. In order to cope with the complexity of the model we use views of the model. A view is a model which is completely derived from another model (the base model). Three views of the NFRs Ontology were identified in [3]: The first view concerns the NFRs relation with the other entities of the software system being developed (intermodel dependency). The second view contains the classes and properties intended to structure NFRs in terms of mutually dependent entities on other NFRs and refinements (intramodel dependency). The third view represents the measurement process and contains the concepts used to produce measures to measurable NFRs. In this paper, we limit the focus to the first two views due to their relevance to the NFRs traceability problem.

A. *Intermodel Dependency View*

Figure 1 illustrates the structure of the NFRs intermodel dependency view by means of a simplified UML class diagram. The core of this structure relies on the fact that NFRs are not stand-alone goals, as their existence is always dependent on other concepts in the project context. If a requirement is a member of the class NonFunctionalRequirement, it is necessary for it to be a member of the class requirement and it is necessary for it to be a member of the anonymous class of things that are linked to at least one member of the class AssociationPoint through the hasAssociationPoint property. On the other hand, isAssociatingNfrTo links the AssociationPoint to a range of: FunctionalRequirement union Element union Process union Product union Resource.

The AssociationPoint can be thought of as an interface from the perspective of the association to the individuals from the above range. Thus, an individual of AssociationPoint class will always associate one or more NFRs to the same one individual from the above range. More specifically:

If an individual is a member of the AssociationPoint class, it is necessary for it to be linked to one and only one individual from: the (FunctionalRequirement class through the isAssociatingNfrTo property) OR (Element through isAssociatingNfrTo property) OR (Process through isAssociatingNfrTo property) OR (Product through isAssociatingNfrTo property) OR (Resource though the isAssociatingNfrTo property).

An individual from AssociationPoint class can be linked to many individuals from the NonFunctionalRequirement class through hasAssociationPoint property.

B. *Intramodel dependency view*

The intramodel dependency view is concerned with the refinement of NFRs into one or more offspring; through either decomposition or operationalization and the correlation among the concepts of the NFRs model. This view is depicted in the simplified UML class diagram in Figure 2.

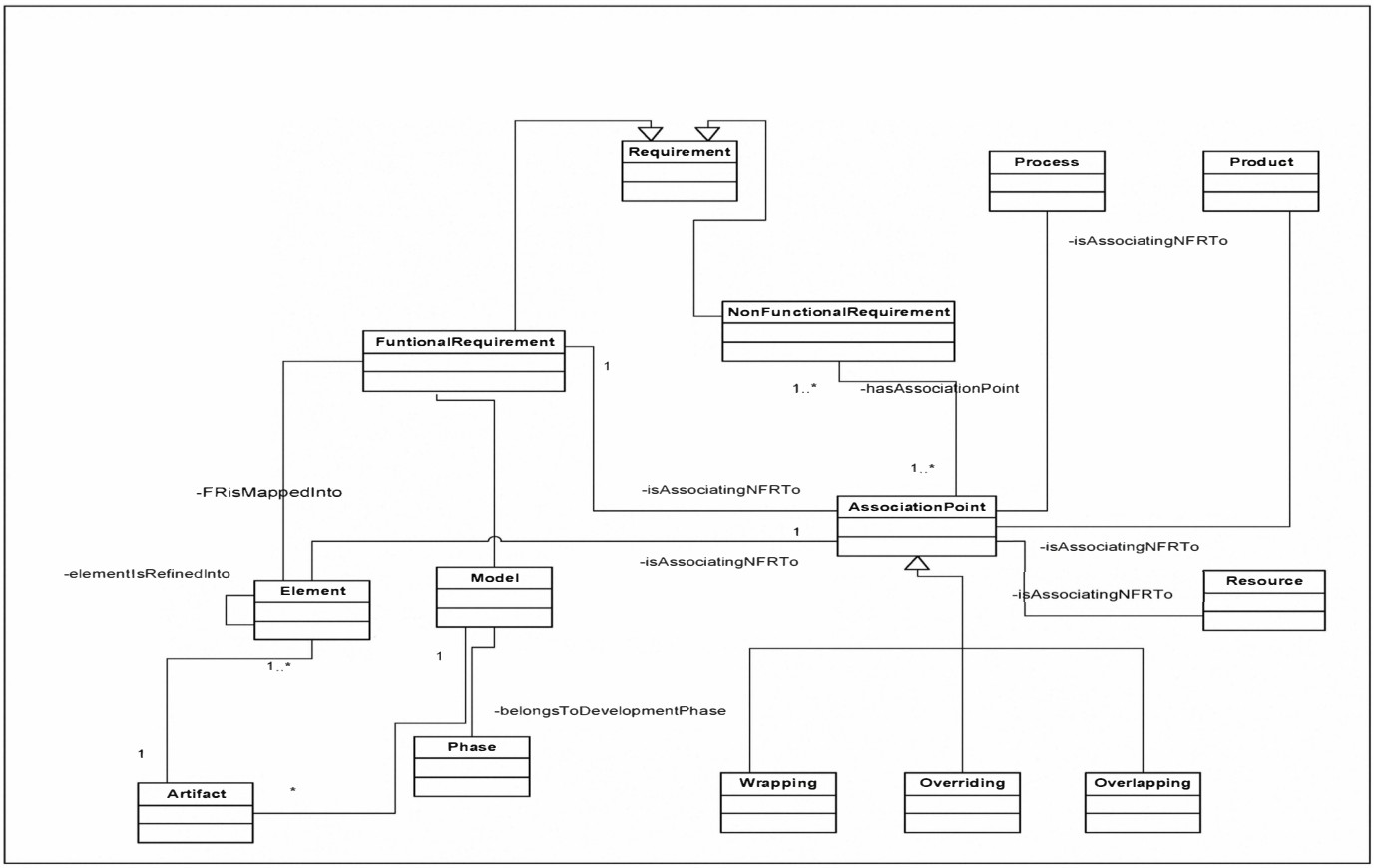

Figure 1. NFRs Intermodel Dependency View.

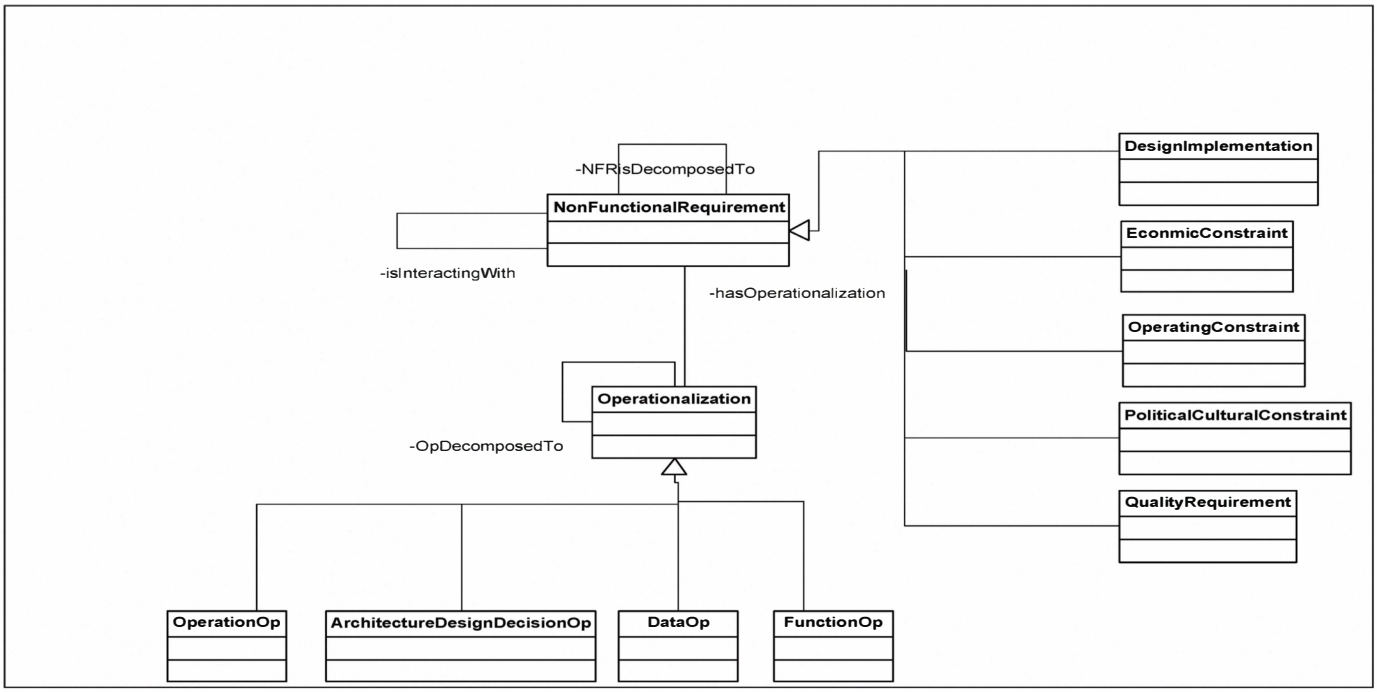

Figure 2 . NFRs Intramodel Dependency View.

Decomposition refers to the NfrIsDecomposedTo property that decomposes a high-level NFR into more specific sub-NFRs. In each decomposition, the offspring NFRs can contribute partially or fully towards satisficing the parent. NfrIsDecomposedTo is a transitive property. 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) [2].

Operationalizations refers to the hasOperationalization property that refines the NFR into solutions in the target system that will satisfice the NFR [2]. An operationalization corresponds to solutions that provide operations, functions (FunctionOp), data representations and architecture design decisions (e.g. design pattern) in the target system to meet the needs stated in the NFRs. Similar to decomposition, operationalization can be ANed or ORed.

On other hand, an individual NFR may participate in isInteractingWith property which links it to another NFR. This refers to the fact that the achievement of one NFR; InfluencerNFR, at a certain association point can hinder (through isNegativelyInteractingWith property) or help (through isPositivelyInteractingWith property) the achievement of other NFR; InfluencedNFR, at the same association point, e.g. security and performance at read an email message functionality. isInteractingWith is not a symmetric property. If NFR1 participates in the relation isNegativelyInteractingWith with NFR2, then we say that there is a conflict between NFR1 and NFR2. A conflict among two or more NFRs occurs when the achievement of one NFR obstructs the achievement of another.

IV. RELATIONAL DATA MODEL FOR TRACING REQUIREMENTS

While the meta models describing the ontology in Figures 1, and 2 are useful ways to understand the abstract structure of the NFRs-related concepts, they are 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 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 [5] 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. In addition, Datalog supports Recursive Closure Operations which makes it possible to trace through multiple levels of refinements within the software development process.

Figure 3 presents the schemas [5] for the relations corresponding to the subset of concepts shown in Figures 1 and 2. The relations are intended to hold information collected by stakeholders at different stages of the development cycle.

We will illustrate the traceability model through examples from the NOKIA mobile email application. The application brings the email experience from recognizable and branded email portals (e.g. Yahoo, MSN, etc.) to the mobile device; and it mirrors the familiar 'look and feel' of the PC, generating instant consumer adoption and virtually eliminating the learning curve.

In this section, we will limit the discussion to two pieces of functionality: (1) the user asks to read an email message; and (2) the user composes and sends a new email. Figure 4 presents a partial view of these two main pieces of functionality decomposed into elements (see Figure 1) of scenarios, messages, and methods. The decomposition of FRs into these elements is for illustrative purposes. Our traceability approach would also support mapping FRs into other refinement elements (e.g. elements of the static view of the system such as classes and relations). Three NFRs are also presented: security, performance, and scalability.

```
//Schema refers to  NonFunctionalRequirement concept
NFR (ID, NAME, DESCRIPTION, SATISFACTION, TYPE);
//Schema refers to  FunctionalRequirement concept
FR (ID, NAME, DESCRIPTION);
//Schema refers to  operationalization concept
OP (OP_ID, NAME, DESCRIPTION);
//Scheme refers to  nfrIsDecomposedTo relation
NFR_DECOMPOSITION (DEC_ID, PARENT_NFR_ID, SUB_NFR_ID,
TYPE_OF_DECOMPOSITION);
/* scheme refers to hasOperationalization relation
(from the NFR to the design solutions) */
NFR_OP (NFR_ID, OP_ID);
//Schema refers to OpDecomposedTo relation
OP_DECOMPOSITION (OP_DEC_ID, PARENT_OP_ID, SUB_OP_ID,
TYPE_OF_DECOMPOSITION);
//Schema refers to isInteractingWith relation
NFR_INTERACTION (INTERACTION_ID,
INTERACTING_ASSOCIATION_ID, AFFECTED_ASSOCIATION_ID,
TYPE_OF_INTERACTION);
//Schema refers to hasAssociationPoint relation
NFR_ASSOCIATION (ASSOCIATION_ID, NFR_ID,
ASSOCIATION_POINT_ID, Type);
//Schema refers to FRisMappedInto relation
FR_ELEMENT (FR_ID, ELEMENT_ID);
//Schema refers to elementIsDecomposedInto relation
ELEMENT_DECOMPOSITION (PARENT_ELEMENT_ID,
CHILD_ELEMENT_ID);
```

Figure 3. Schematic representation of some concepts and relations presented in Figures 1 and 2.

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 participating in hasAssociationPoint relation with individual from AssociationPoint class which in its turn participates in isAssociatingNfrTo relation with the individual "read an email message" instantiated from FunctionalRequirement class. Confidentiality, which is derived from security by "ANed" decomposition (through NfrIsDecomposedTo relation), also participates in hasAssociationPoint relation with the same individual from AssociationPoint class which participates in its turn in isAssociatingNfrTo relation with "read an email message". This information on confidentiality association could be missed when populating the NFR_ASSOCIATION relation, yet this relation has to be traced on 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 subsections.

A. Impact of Changes to Functional Models on NFRs

When a change is initiated in an FR, the set of NFRs potentially affected needs to be identified and retrieved. This is accomplished by first retrieving all the directly associated NFRs from the relation NFR_ASSOCIATION. In order to ensure the completeness of the trace and the consistency among requirements, it is important that all NFRs associated with all elements derived from the affected FR against the requested change be analyzed as well. This should be done in a recursive manner to cover all possible derived elements. The following Datalog expressions implement this query:

// R_TEMP refers to a temporary relation.
/* FR_CHANGED and NFR_CHANGED refer to the ID of the FR and the NFR, the 'request changes' from which the need for traceability was triggered. */
/* RESULT refers to the desired relation that holds the data result. */
R1_TEMP(Y) ← FR_ELEMENT(X,Y) , X = "FR_CHANGED"
R2_TEMP (Q, W) ← ELEMENT_DECOMPOSITION (Q, W), R1_TEMP (Y), Q = Y
R2_TEMP (Q, W) ← ELEMENT_DECOMPOSITION (Q, Z) , R2_TEMP (Z, W)
RESULT (B) ← NFR_ASSOCIATION (A, B, C, D) , C = "FR_CHANGED"
RESULT(B) ← NFR_ASSOCIATION (A, B, C, D), R2_TEMP (Q, W), C= Q
RESULT(B) ← NFR_ASSOCIATION (A, B, C, D), R2_TEMP (Q, W), C= W

It is important to note that the decomposition of NFRs will never have a circular dependency. This is a necessary condition for the termination of R2_TEMP. In the case study of the mobile email system (see Figure 4), if a change is requested to the read an email message functionality, then the above query expressions will retrieve security, performance, and scalability as potentially impacted NFRs.

B. Impact of Changes to Nonfunctional Models on Functional Models

To ensure a complete inter-model traceability, we should consider the impact of changes to NFRs on the functional model to complement the query in Subsection A which considered the impact of changes of functional models to NFRs. When a change is initiated in an NFR, then the set of all association points of the FR type or of the element type should be retrieved and analyzed against the potential change. The following Datalog expressions implement this query:

RESULT(B) ← NFR_ASSOCIATION (A, B, C, D), D = "FR", B = "NFR_CHANGED".

RESULT(B) ← NFR_ASSOCIATION (A, B, C, D), D = "ELEMENT", B = "NFR_CHANGED".

In the mobile email system (see Figure 4), security could be known as being participating in *hasAssociationPoint* relation with individual from *AssociationPoint* class which in its turn participates in *isAssociatingNfrTo* relation with the individual "*read an email message"* instantiated from *FunctionalRequirement* class. If a change is requested to a security requirement, then the above query expression will retrieve the read an email message functionality, all derived success and alternate scenarios, and the corresponding elements such as select a message and open the selected message, as well as the methods m1, m2, m3 and m4.

C. 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 expression implements this query:

TEMP_1 (B,C) ← NFR_DECOMPOSITION (A, B, C, D), B = (NFR_CHANGED)
TEMP_1 (B,C) ← NFR_DECOMPOSITION (A, B, C, D), C = (NFR_CHANGED)
TEMP_1 (B, C) ← NFR_DECOMPOSITION (A, B, C, D), TEMP_1 (X, B)
RESULT (X) = TEMP_1(X, Y), X <> (NFR_CHANGED)
RESULT(Y) = TEMP_1(X,Y), Y <> (NFR_CHANGED)

In the mobile 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.

D. 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:

RESULT(Y) ← NFR_INTERACTION (X,Y,Z,W), Z = "CHANGED_NFR".
RESULT(Z) ← NFR_INTERACTION (X,Y,Z,W) , Y= "CHANGED_NFR".

In the mobile 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.

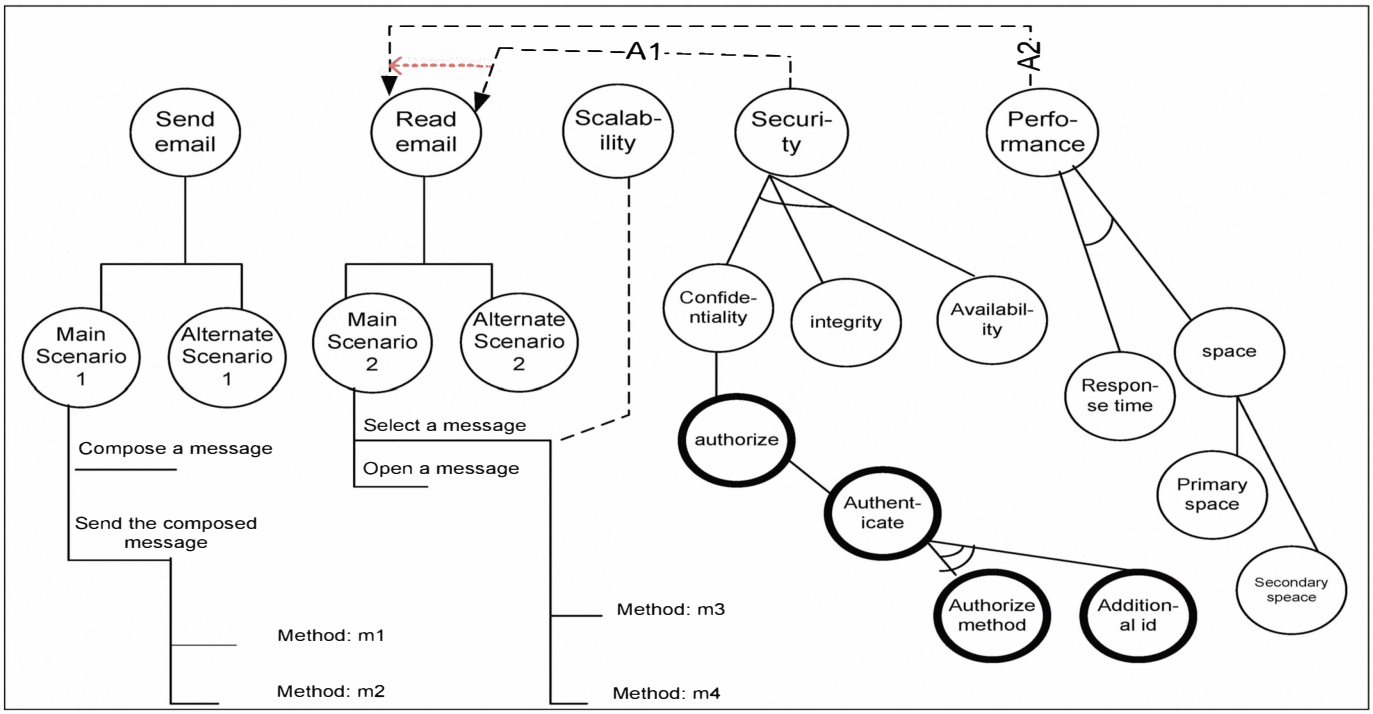

Figure 4: Illustration of FR and NFR Relations through the Mobile Email System.

E. Guidelines for populating the traceability realtional model

Below, we restate the applicability of the traceability approach with steps towards deploying the approach in practice.

1- Transform manually the NFRs Ontology into corresponding relational-model based representation.
2- Upon a change request, identify the potentially impacted areas along with their specifications and refinements.
3- Execute the corresponding query.
4- Once the retrieval algorithm has returned a set of potentially impacted requirements / elements, filter the retrieved requirements/elements to remove any non-relevant ones.
5- A decision on any accepted change in any of the retrieved data should be recorded in the corresponding relations.

V. Improving Regression Testing with the Traceability Mechanism

Testing represents a major effort within the software development cycle. The Guide to the Software Engineering Body of Knowledge (SWEBOK) [21] provides an overview of the basic and generally accepted notions underlying the software testing knowledge area.. Testing implies a trade-off between limited resources and schedules, and inherently unlimited test requirements. As a result, one needs a finite test set with which enough testing is conducted to obtain reasonable assurance of acceptable behavior and quality.

We note that test-cases generation is out of the scope of this paper; instead, we aim at improving the quality of regression testing by applying our traceability approach. Regression testing means rerunning test-cases from existing test suites to build confidence that software changes have no unintended side-effects [22]. The "ideal" process would be to create an extensive test suite and run it after each and every change. For many projects this is nearly impossible because test suites are too large or because changes come in too fast [22]. Researchers have tried to make regression testing more effective and efficient by developing regression test selection techniques, but many problems remain, such as limited resources or testing time. These and other issues have not been adequately considered in current research, yet they strongly affect the applicability of proposed regression testing processes. Moreover, regression testing in the industry is mostly based on experience and not on systematic approaches [23]. We believe that the traceability mechanism proposed in this paper can be exploited to dramatically improve the costs and benefits of the regression testing in the industrial context. Because Regression testing seeks to uncover new errors, or regressions, in existing functionality after changes have been made to the software, this type of testing is concerned with changes to existing features rather than the new introduced features and thus it was selected as a fit to illustrate the proposed change management approach.

A. Research Hypothesis

To meet the need for high-quality regression testing, we propose to integrate the traceability mechanism within the regression testing activity. Our research hypothesis states the following: "Applying the traceability mechanism proposed in this paper into the regression testing will improve the effectiveness and efficiency of the regression testing suite; that is, for a lower number of test-cases to be executed within a given amount of time, a higher number of defects will be detected".

B. Applying the traceability mechanism to test-cases selection

For the purpose of the evaluation of the traceability approach, we used the settings from the NOKIA Mobile Email

Application System to run a multi project variation quasi-experiment [24].

The NOKIA mobile email application is deployed on hundreds of branded cell phones. Change requests are received from the email providers, operators or upon a defect discovery. As a testing practice in NOKIA, upon triggered changes in the requirements, the fix procedure starts and it involves a sanity testing activity. Sanity test is a brief run-through of the functionality of the software system to assure that the system works as expected. The activity is carried on by an execution of a fixed set of regression test-cases (25 test-cases out of more than 10,000 implemented test-cases) to check that the implemented changes didn't break other features. Of course, the small number of test-cases (25) is due to limitation of time and available human-resources. The fixed set of sanity test-cases are pre-selected manually by the QA manager and they cover a set of functionalities which are deemed to have the highest priority for the client.

In this quasi-experiment, first, we link the requirements and the design solutions with their corresponding test-cases in the test management software. That is, each identified test-case has to be linked to at least one requirement or design solution. Second, upon a change request classified manually as one of the identified critical areas (see Section IV of this paper), the potentially affected requirements and design solutions have to be retrieved by executing the recommended queries. Third, the corresponding test-cases which are linked to the retrieved requirements and design solutions will be selected from the test-cases database. This is of course in addition to the test-cases which are directly linked to the requirement which is referred to by the requested change.

C. Evaluation and demonstration of regression testing improvement due to traceability queries

The objective of the quasi-experiment was to evaluate the research hypothesis stated above. The set of dynamically generated test-cases from applying the traceability mechanism was executed in addition and in isolation of the fixed set of sanity test-cases. The results were then compared. This quasi-experiment was carried out by the same team of client testers at NOKIA-Montreal office on 40 mobile email projects for a period of nine months from July 2008 till March 2009. The number of the dynamically generated test-cases to be executed varied in each run depending on the triggered change.

To understand the improvements, which the use of traceability queries brings to the testing practice, we compare the practice of using the fixed set of sanity test-cases against the practice of using dynamically generated test-cases with the help of our traceability queries. Each failed test-case prompts the tester to create a defect.

The average number of defects being discovered per sanity-test execution using the dynamically generated test-cases method over the 40 executions we performed is 1.825, while it is 0.775 using the fixed set of sanity test-cases (see Figure 6). This is an increase of 235%. In addition, the average number of dynamically generated test-cases being executed was 19.25 over the 40 executions. That is less by 33% from the fixed set of 25 test-cases (see Figure 5). These results demonstrate empirically that the stated hypothesis that the traceability queries were useful in improving the effectiveness and efficiency of the regression testing practice.

VI. CONCLUSION AND FUTURE WORK

The tendency for NFRs to have a wide-ranging impact on a software system, and the strong interdependencies and tradeoffs that exist between NFRs and the software architecture, leave typical existing traceability methods incapable of tracing them. In this paper, we use the NFRs Ontology specification for requirement relations in a real life industrial setting. We proposed and deployed a traceability mechanism under the umbrella of the relational model and the Data log expressions 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. A major limitation of the proposed approach is related to stakeholders and their ability to identify all relevant relations. 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 immediately reachable from that NFR within the hierarchy. The approach has been evaluated and demonstrated its applicability through a multi project variation quasi-experiment performed against the Mobile Email application in NOKIA-Montreal. The replicated quasi-experiment presented in section V is an initial evaluation to the proposed traceability mechanism for NFRs.

We believe that benefits which arise by blending our research results with existing industry practice can further make an enhancement of their experience about requirements traceability. For example a valid traceability approach will allow the industry to improve the synergies among their RE, architectural design, implementation and testing processes

To collect more evidence about the merits of our approach and better judge its validity, we plan further evaluation studies on the traceability mechanism that include extending its applicability beyond the testing activities (e.g. requirements review activities, project's extension.) This will be done by applying empirical research methods, specifically case studies and experiments [25].

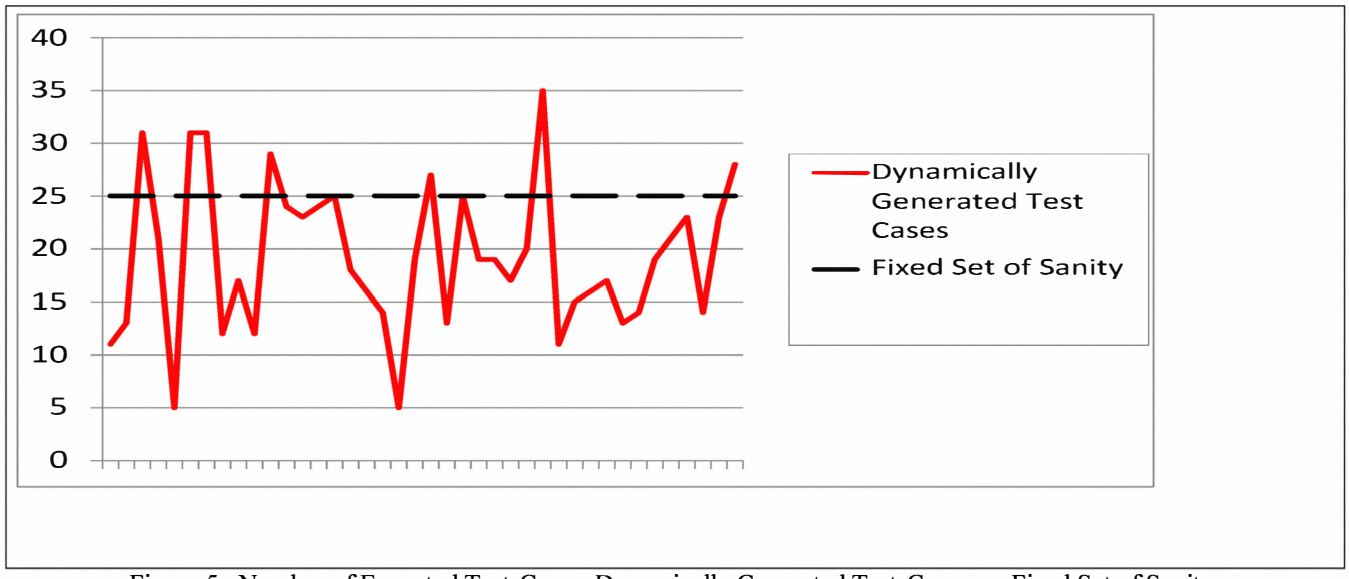

Figure 5: Number of Executed Test-Cases: Dynamically Generated Test-Cases vs. Fixed Set of Sanity.

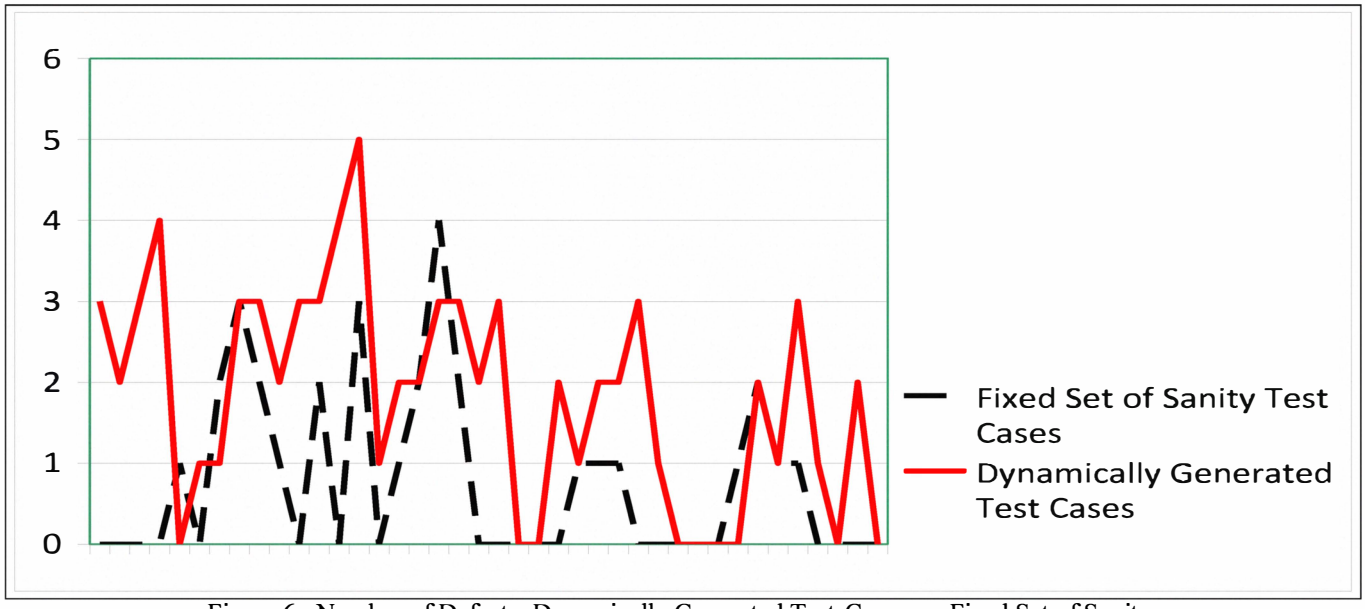

Figure 6: Number of Defects: Dynamically Generated Test-Cases vs. Fixed Set of Sanity.

REFERENCES

[1] M. Weber and J. Wesbrot, "Requirements Engineering in Automotive Development: Experiences and Challenges", IEEE Software, vol. 20 (1), pp.16-24, 2003.

[2] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos, Nonfunctional Requirements in Software Engineering, Kluwer Academic Publishing, 2000.

[3] M. Kassab, O. Ormandjieva, and M. Daneva, "An Ontology Based Approach to Non-Functional Requirements Conceptualization", Proceedings of the Fourth International Conference on Software Engineering Advances (ICSEA 2009), Porto, Portugal, pp. 299-308, 2009.

[4] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, and S. Christina, "Goal Centric Traceability for Managing Non-Functional Requirements", Proceedings of the 27th international conference on Software engineering, pp. 362 – 371, 2005.

[5] J. Ullman and J. Widom, Database Systems: The Complete Book, Prentice Hall, 2000

[6] L. Chung, B.A. Nixon, and E. Yu, "Using Non-Functional Requirements to Systematically Support Change", Proceedings of the Second IEEE International Symposium on Requirements Engineering, York, U.K., pp. 132 – 139, 1995.

[7] J. Cleland-Huang, "Toward Improved Traceability of Non-Functional requirements", Proceedings of the 3rd international workshop on Traceability

in emerging forms of software engineering, Long Beach, California, pp. 14 – 19, 2005.

[8] A. Egyed and P. Grunbacher, "Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help", IEEE Software, vol. 21(6), pp. 50- 58, 2004.

[9] A. Finkelstein and W. Emmerich, The Future of Requirements Management Tools, In Information Systems in Public Administration and Law, G. Quirchmayr, R. Wagner and M. Wimmer (Eds.): Oesterreichische Computer Gesellschaft, 2000.

[10] A. M. Salem, "Improving Software Quality through Requirements Traceability Models", Proceedings of International Conference on Computer Systems and Applications, pp. 1159- 1162, 2006.

[11] B. Ramesh and M. Jarke, "Toward a Reference Model for Requirements Traceability", IEEE Transactions on Software Engineering, vol. 27(1), pp. 58-93, 2001.

[12] C. Hofmeister, R.L. Nord, and D. Soni, "Global Analysis: moving from software requirements specification to structural views of the software architecture", IEE Proceedings Software, vol. 152(4), pp.187- 197, 2005.

[13] D. Jacobs, "Requirements Engineering: so Things Don't Get Ugly", Companion to the Proceeding of 29th International Conference on Software Engineering, pp. 159- 160, 2007.

[14] E. Baniassad, P. C. Clements, J. Araujo, A. Moreira, A. Rashid, and B. Tekinerdogan, "Discovering Early Aspects", IEEE Software, vol. 23(1), pp. 61- 70, 2006.

[15] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, Addison-Wesley, NY, 2003.

[16] E. Niemelä and A. Immonen, "Capturing Quality Requirements of Product Family Architecture, Information and Software Technology", vol. 49(11- 12), pp. 1107-1120, 2007.

[17] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni, "Model Traceability", IBM System Journal, vol. 45(3), pp. 515- 526, 2006.

[18] O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem", Proceeding First International Conference Requirements Engineering, Colorado, U.S.A, pp. 94-101, 1994.

[19] P. Letelier, "A Framework for Requirements Traceability in UML-Based Projects", Proceeding of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering, Edinburgh, pp. 30–41, 2002.

[20] V. Winter, H. Siy, M. Zand, and P. Aryal, Early Aspects Workshop at AOSD'06, Bonn, Germany, 2006.

[21] A. Bertolino, "Knowledge area description of software testing guide to the SWEBOK", Available at: http://www.swebok.org , 2004.

[22] J. M. Kim, A. Porter and G. Rothermel, "An Empirical Study of Regression Test Application Frequency", The Journal of Software Testing, Verification & Reliability, vol. 15 (4), pp. 257-279, 2005.

[23] E. Engström and P. Runeson, "A Qualitative Survey of Regression Testing Practices", In Product-Focused Software Process Improvement, Lecture Notes in Computer Science, 6156, pp. 3-16, Springer Berlin / Heidelberg, 2010.

[24] V. R. Basili, "The role of experimentation in software engineering: past, current, and future", Proceedings of the 18th international conference on Software engineering, Berlin, Germany, pp. 442- 449, 1996.

[25] C. Wohlin, Experimentation in Software Engineering, Springer, 2001.