Toward Improved Traceability of Non-Functional Requirements

Jane Cleland-Huang
Center for Requirements Engineering
School of Computer Science, Telecommunications, and Information Systems
DePaul University
Chicago, IL. USA

jhuang@cs.depaul.edu

ABSTRACT

This position paper examines current practices and challenges for tracing non-functional requirements (NFRs). Anecdotal evidence suggests that many organizations do not effectively trace NFRs and that functional changes are often implemented with very little understanding as to how system qualities such as safety, security, and performance will be impacted. The tendency for NFRs to have broad ranging impact upon a software system, and the strong interdependencies and tradeoffs that exist between NFRs and the software architecture leave typical existing traceability methods insufficient for tracing NFRs. This paper focuses on post-requirements traceability of NFRs. It first identifies three critical areas in which NFRs require traceability support, it then evaluates existing traceability methods, and finally proposes a more holistic traceability environment named Goal Centric Traceability that supports impact analysis of NFRs within the context of the software architecture in which they are deployed.

Categories and Subject Descriptors

D.2.1 Requirements/Specifications, D.2.2 Design Tools and Techniques,D.2.11 Software Architectures,D.2.13 Reusable Software.

General Terms
Documentation, Management, Design

Keywords
Traceability, Software Architecture, Non-functional requirements.

1. WHY TRACE NFRS ANYWAY?
Requirements traceability provides critical support for many essential software development activities. In most non-trivial projects traceability is used to model dependencies between requirements, to track the allocation of requirements to system components, manage compliance verification, and control changes to the system. Traceability helps the developer understand the relationships that exist within and across software requirements, design, and implementation [10].

In practice, many organizations either focus their traceability efforts on functional requirements or else entirely fail to implement an effective traceability process. In many organizations non-functional requirements (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. However with appropriate tool support NFR traceability 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 problems associated with inadequate NFR traceability are well illustrated through a rather typical incident experienced in the late 1990s by a large U.S telecommunications company. The Japanese government introduced legislation requiring all telephone service providers to support a new international dialing prefix and allow customers to use a 5-digit carrier access code. While implementing the new feature, analysts and developers focused upon the system functionality of the mobile switching center and failed to consider the impact of the change on system performance. When the upgrade was deployed, it caused several critical processors to crash, resulting in customer dissatisfaction and frantic efforts to get the system back on line. A postmortem analysis determined that the analysts had failed to consider the impact of functional change on non-functional system qualities. Despite the fact that existing simulations could have been used to assess the impact of the change, there had been no way for developers to identify applicable ones [5].

David Parnas gave another example of poor traceability in a safety critical system [18]. While visiting a major U.S. airport he noticed a red alarm lamp on the air traffic controller's desk and discovered that nobody had any real idea what its purpose was. The supervisor pointed him to the system documentation, which took up a considerable amount of shelf space and would have taken several weeks to read! In addition, modifications to the original specifications were stored as separate reports, so that in fact the specifications for triggering the alarm light were distributed in several documents. The traceability between potentially safety critical functionality and supporting documents had been entirely lost over the years and changes in the system made documentation too antiquated to be any use at all.

Anecdotal accounts from industry continue to indicate a failure to adequately trace NFRs suggesting that numerous organizations are exposing themselves to enormous and unnecessary risks.

Improving requirements traceability is particularly important to the problem of handling change in non-functional requirements because their failure can lead to catastrophic results with far reaching impact.

2. NFR TRACEABILITY CHALLENGES

Although most prior work on requirements traceability does not explicitly differentiate between functional and non-functional requirements there are some clear differences in the way impact analysis and traceability of NFRs must be performed. These differences originate primarily from the tendency for NFRs to have a global impact upon a software system, and secondly from the strategic role played by NFRs in driving the architectural design of a system. The relationship between NFRs and architecture has been noted by several researchers, including Nuseibeh, who illustrated the need to iteratively and concurrently define quality attribute requirements alongside the architectural design of the system [16].

Impact analysis of NFRs must therefore first identify which NFRs are potentially impacted by a change, and secondly determine the extent to which that change adversely impacts architectural quality. Identifying the initial impact of a change upon system wide NFRs represents a model-to-model traceability problem and is initiated either by a change in functional requirements or in downstream artifacts such as code or design.

The actual process used to evaluate the impact of a change against the software architecture is highly dependent upon the type of NFR under evaluation. Qualities such as *maintenance* and *extensibility* tend to impact the system at the design level and so traceability support is needed between NFRs and design.

Other types of NFR such as *safety*, *security*, and *performance* are more complex to trace. In addition to impacting the software design, these qualities have a strong runtime impact and are affected by factors such as hardware configuration and usage patterns. Over the long term these NFRs need to be evaluated not only in respect to the design, but in respect to the runtime environment. Traceability links must support the evaluation of the system's long-term compliance to its quality goals through defining relationships between NFRs and strategically placed runtime monitors or other types of captured metrics.

Architectural assessment models (AAMs) constructed during initial system design can provide useful support for impact analysis of NFRs. These models could for example include system execution models to test for performance, anti-scenarios constructed to test the security of the system[13,4], or more complex models such as hybrid automaton used to demonstrate that a design conforms to stated safety requirements[5]. In typical practice these models are used and then discarded or 'lost'. However effective NFR traceability practices should establish traces between AAMs and NFRs in order to make these models available for reuse throughout the lifetime of the software system.

During early system analysis, quality attributes tend to be initially defined in terms of high-level and somewhat fuzzy goals and then refined into lower level goals for which candidate architectural solutions can be proposed and evaluated. AAMs may be developed to evaluate goals at any level of the hierarchy. If the impact of a change is detected upon a lower level goal, then vertical traces (also known as intra-model traces) must be utilized to traverse the goal tree and identify higher level impacted goals and their related AAMs.

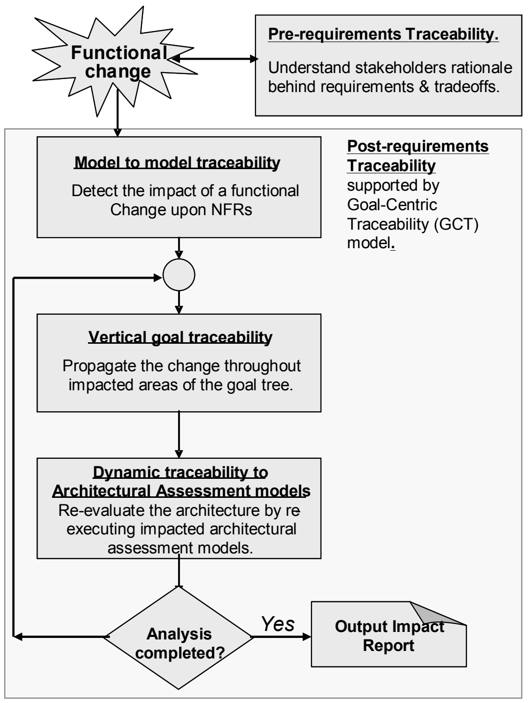

Functional change
Pre-requirements Traceability.
Understand stakeholders rationale behind requirements & tradeoffs.
Post-requirements Traceability supported by Goal-Centric Traceability (GCT) model.
Model to model traceability
Detect the impact of a functional Change upon NFRs
Vertical goal traceability
Propagate the change throughout impacted areas of the goal tree.
Dynamic traceability to Architectural Assessment models
Re-evaluate the architecture by re-executing impacted architectural assessment models.
Analysis completed?
Yes
Output Impact Report

Figure 1. Traceability Elements

This position paper proposes three distinct traceability elements needed to support the impact analysis of NFRs. These are modeled in Figure 1 and include the following:

• **Impact detection** (model-to-model traceability) in which bidirectional traceability is established between the functional and non-functional models of the system. When a change is initiated in either the requirements specification or in the design the set of potentially impacted NFRs can be identified and retrieved.

• **Traceability between NFRs and Goals** (intra-model traceability) in which the impact of a change in a lower level operationalization or goal is propagated to higher level goals. This type of traceability enables an analyst to understand the impact of a low level change on high level goals and vice versa.

• **Dynamic traceability of architectural assessment models** (model-to-model traceability) in which traces that have been explicitly established between goals and architectural assessment models (such as simulations and other executable models) are used to re-evaluate the architecture's ability to accommodate the proposed change.

3. EXISTING TRACEABILITY METHODS

Although prior work on tracing NFRs has been rather limited, a number of traceability techniques have in fact been developed to support related activities such as architectural assessment, aspect weaving, and test set selection. The following section briefly surveys several of these methods, and discusses their strengths and weaknesses for supporting the three proposed elements of NFR traceability. The methods are summarized in Table 1.

3.1 Matrices

Traceability matrices are commonly used in industry to define relationships between requirements and other types of artifacts

Table 1. Evaluation of selected NFR traceability methods

Traceability Technique	Support for: Trace creation & maintenance	Support for: Identifying impacted artifacts	Support for: Evaluating the scope of the impact	Weaknesses	Strengths
Matrices	Manual	Automated or manual	Could automate basic metrics.	Problems with scalability and long term maintenance lead to accuracy problems.	Simple. Well accepted in industry.
Keywords	Manual	Automated	Manual	Problems with scalability and long term maintenance.	Traceability logic is distributed into the artifacts. Keywords provide visual clues to traceable artifacts.
Aspect weaving	Automated once rules are defined.	Automated (Aspect weaver)	No support	Only applicable for NFRs that can be represented as aspects.	Links are dynamically generated. Concept could be explored outside an aspect oriented environment.
IR methods	Automated	Semi-automated	Manual	Recall and precision of results is not 100%. User input required to validate results.	Candidate links are dynamically generated.
Scenario based	Manual	Automated	Manual / Automated	Scenarios unlikely to provide complete traceability coverage.	Scenarios are multi-purpose requiring less overhead to establish traceability.
Event based	Manual	Automated	Automated	Only a subset of AAMs support the the API required by EBT.	Supports automated re-evaluation of simulations, monitors, and other types of architectural assessment models.
Process centered	Modeled & enforced by PCEE	Semi-automated	Manual	Requires fine-grained definition of method fragments. High-overhead.	Processes could be defined to support traceability to AAMs.
Design Pattern based	Manual	Semi-automated	Manual	Pattern detection techniques return imprecise results.Certain patterns easier to detect.	Provides a method for tracing NFRs to design without resorting to a proliferation of links.

such as code modules, design, and test cases. Although matrices are technically capable of supporting NFR traceability, they require links to be explicitly defined and maintained and do not tend to scale well for the prolific number of links that can be generated from a single NFR. In industry their use tends to be limited to the definition of a few critical NFRs. Many traceability features that are part of requirements management tools unfortunately also suffer from the same maintenance problems as matrices.

3.2 Keywords and Ontology

Cysneiro et al described a method for supporting traceability between UML diagrams and NFRs modeled in a goal tree through use of a Language Extended Lexicon (LEL) [9]. Keywords representing both domain and non-functional qualities are embedded into the NFR goal tree and into UML diagrams and used to establish traceable relationships. This approach eliminates the need to maintain a central traceability matrix but necessitates the discipline of constructing and conforming to a set of reserved keywords and ensuring that they are maintained and used systematically throughout the evolution of the system.

3.3 Aspect Weaving

Recent interest in aspect oriented programming (AOP) has focused attention on the problem of cross-cutting concerns that result in similar functionality dispersed broadly across a system [3]. In AOP, suitable concerns are modeled as aspects in which the dispersed functionality is encapsulated into a single entity and woven into the code by a special compiler according to a set of aspect weaving rules. Concerns are categorized as functional or non-functional. Non-functional concerns include high-level NFRs such as *maintainability*, *performance*, and *security* etc and tend to be insufficiently concrete to be implemented as aspects. On the other hand, functional and more concrete concerns such as *logging*, *authentication*, and *tracing* can be implemented as aspects because their behavior is more easily definable and can be expressed in terms of aspect weaving rules. From a requirements perspective many of the aspects labeled 'functional' are in fact lower level refinements of higher level NFRs.

AOP therefore establishes traceability between aspects (representing low level NFRs) and code. However the approach requires a special AOP compiler and is only appropriate for the subset of NFRs that can be implemented as aspects.

3.4 Information Retrieval Methods

In direct response to problems experienced by practitioners in manually creating and maintaining traceability links, many researchers have recently investigated the use of information retrieval (IR) methods to dynamically generate traces [1,8,12,15]. IR methods generally calculate a similarity score based on the frequency and dispersion of terms in both the query and searchable documents. Candidate links are returned for pairs of artifacts whose similarity score is higher than a certain threshold or who fall in the top percentage of potential links. Most reported results have returned precision of between 10-50% while recall levels are fixed at 90%. Although results are influenced by the algorithm used and can be improved by the use of feedback mechanisms and incorporation of additional contextual knowledge, the real limitation appears to be related to the quality of the actual dataset. Datasets that use clearly defined project glossaries tend to return better results than more haphazardly defined ones.

Although much of this research has focused on tracing functional requirements Cleland-Huang et al utilized a probabilistic retrieval algorithm to trace non-functional requirements [8]. Traces were dynamically generated between UML class diagrams and elements of a Softgoal Interdependency Graph (SIG). These links were designed to support an analyst in understanding the impact of design changes upon non-functional systemic goals. At fixed recall levels of approximately 90%, precision of 20% to 50% was achieved for different types of NFRs.

3.5 Scenario based traceability

Scenarios are commonly used to model system functionality and to drive the generation of functional test cases. Scenario based test-cases create a mapping between requirements and other artifacts such as design and code. Several architectural assessment methods such as the Architectural Tradeoff Assessment Method (ATAM) [14] and the Software Architecture Assessment Method (SAAM) [13] also utilize scenarios and anti-scenarios (i.e. scenarios that attempt to harm or break the system) to evaluate the quality of the architecture in respect to its NFRs. For example, a security goal to allow only authenticated access could be tested by developing an anti-scenario in which a malicious user attempts to illegally access the system. Explicit traceability is established through the mapping of the scenario onto design elements. There is however no guarantee that the scenarios will provide complete coverage as they are typically developed to explore only a subset of interesting cases. In most cases scenarios also need to be created and manually maintained.

Use case maps further enhance the traceability functionality of scenarios as they provide an explicit notation for mapping a scenario onto an architecture [2].

3.6 Event-based Traceability

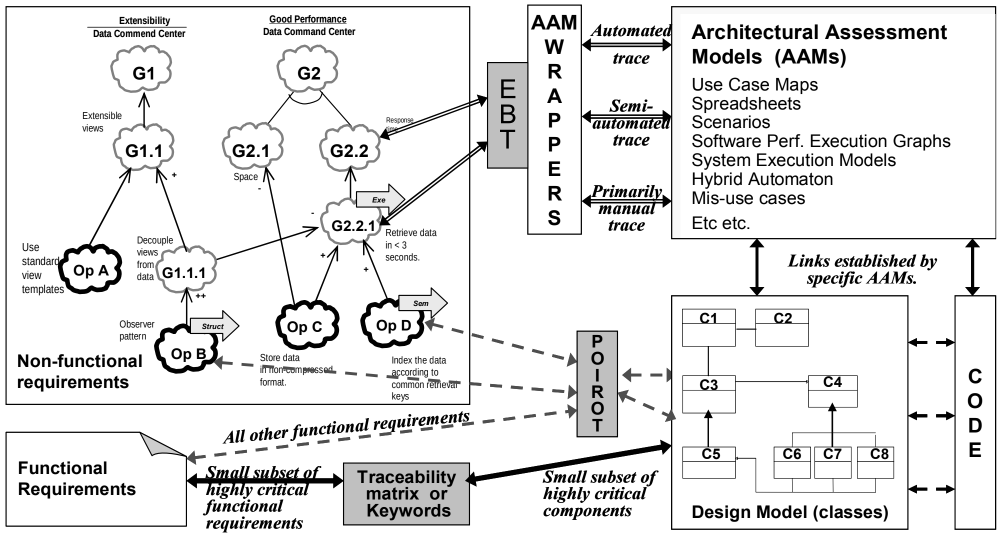
Event-based traceability utilizes a publish-subscribe architecture to define traceability links that support automated impact analysis of NFRs [5]. Links are established between quantitatively defined performance requirements, constraints in the requirements specification, and variables embedded in performance simulation models. When a change is proposed the current state of the simulation model is saved, speculative parameters are passed to the model, the model is re-executed using the new values, results are extracted and reported, and the model is restored to its initial state. EBT therefore demonstrates the ability to automate the re-execution of an architectural assessment model in response to a change once an impacted requirement has been discovered, however prior work in this area has focused primarily on performance modeling and evaluation.

3.7 Process Centered Environments

Pohl et al established traceability through implementing a process centered engineering environment (PCEE)[19]. A PCEE is composed of the three domains of modeling, enactment, and performance. The modeling domain defines process and traceability tasks; the enactment domain enforces, guides, and controls the software engineering process and related traceability tasks; while the performance domain represents the actual implementation of those tasks by a human or a computer system. Although examples provided [19] focus on tracing functional requirements, this approach could potentially be applied to tracing non-functional requirements by establishing and utilizing method fragments to connect NFRs and architectural assessment models within the managed enactment domains.

3.8 Design Patterns

Gross and Yu proposed the use of design patterns for establishing traces between non-functional goals in a goal tree such as a softgoal interdependency graph (SIG) and the system design [11]. Cleland-Huang et al expanded on this idea through defining a model that utilized design patterns as intermediate trace artifacts between a SIG and the underlying OO design [6]. The approach was based on the application of pattern detection algorithms,



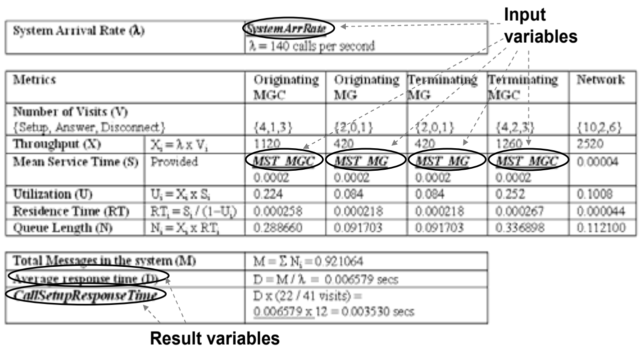
Dynamically generated links.
Modeled here using our tool Poirot that implements a probabilistic network algorithm.

Structural links.
Inherent to the model, for example links between goals in the goal tree or between classes in the design model

Executable links.
Links that can be used to invoke executable code. Modeled here using Event Based Traceability (EBT).

Traditional (manually constructed/non executable) links
Typically supported by a traceability matrix or use of keywords.

Figure 2. GCT model for tracing Non-Functional Requirements



System Arrival Rate (λ)	*SystemArrRate* λ = 140 calls per second	

Metrics		Originating MGC	Originating MG	Terminating MG	Terminating MGC	Network
Number of Visits (V) {Setup, Answer, Disconnect}		{4,1,3}	{2,0,1}	{2,0,1}	{4,2,3}	{10,2,6}
Throughput (X)	$X_i = \lambda \times V_i$	1120	420	420	1260	2520
Mean Service Time (S)	Provided	*MST_MGC* 0.0002	*MST_MG* 0.0002	*MST_MG* 0.0002	*MST_MGC* 0.0002	0.00004
Utilization (U)	$U_i = X_i \times S_i$	0.224	0.084	0.084	0.252	0.1008
Residence Time (RT)	$RT_i = S_i / (1-U_i)$	0.000258	0.000218	0.000218	0.000267	0.000044
Queue Length (N)	$N_i = X_i \times RT_i$	0.288660	0.091703	0.091703	0.336898	0.112100

Total Messages in the system (M)	$M = \Sigma N_i = 0.921064$
Average response time (D)	$D = M / \lambda$ = 0.006579 secs
CallSetupResponseTime	D x (22 / 41 visits) = 0.006579 x 12 = 0.003530 secs

Figure 3. A Performance AAM [5]

within a subset of high-level explicitly traced classes. The technique supports traceability for any NFR that can be implemented as a design pattern. However current pattern detection algorithms work better for certain design patterns than others and in their current form tend to return additional unwanted links

4. GOAL CENTRIC TRACEABILITY

This position paper has documented several methods that are currently used in industry to trace NFRs or have been proposed as research prototypes. However none of these techniques provide sufficient coverage for supporting the three essential traceability elements needed by NFRs and defined earlier in section 1 of this paper.

We therefore propose a more holistic approach called Goal Centric Traceability. Some of the ideas in GCT were introduced in an earlier paper [8] however the model is now expanded to incorporate model-to-model traceability between goals and architectural assessment methods in order to provide traceability support for performing NFR related impact analysis within the context of the deployed software architecture.

4.1 Features of Goal Centric Traceability

The GCT model, which is illustrated in Figure 2, incorporates the following features:

• **Goal oriented decomposition** A goal oriented approach supports both formal and informal software development. The goal tree provides explicit traceability between higher level goals and lower level architectural features and their associated functional requirements. When a change impact is detected upon a lower level node in the goal tree, the impact analysis can be propagated to other indirectly related goals in the goal tree.

• **Explicit traces to architectural assessment models** Architectural assessment models are traced to NFR goals through the use of event based traceability (EBT). GCT improves on existing EBT techniques by establishing links within the context of the goal tree. This provides connectivity between multiple models related though the goal tree, and enables the output of one model to be fed as an argument into another model.

• **Automated Support for Impact Analysis** Estimating the impact of change upon NFRs is much harder than estimating the impact upon functional requirements. Impact analysis is concerned with understanding the scope, depth, and potential side-effects of a proposed change. In GCT traceability links are established between goals, architectural assessment models (AAMs), simulations, and runtime monitors etc. During impact analysis these links are used dynamically to re-execute these models or poll runtime monitors. During a change event the GCT objective is to identify potentially impacted AAMs and simulations etc, to re-evaluate them within the context of the changed requirements or changed environmental context, and to return a result that is meaningful for evaluating the satisficing of their related NFR goal. This process is partially supported through EBT however it is necessary to construct additional wrappers and APIs to interface a broader spectrum of AAMs with the GCT model so that they can be incorporated into the automated impact analysis.

• **Explicit traceability for critical requirements** GCT uses both explicit traceability and dynamically generated methods according to the criticality of the requirement being traced. The imprecision of existing dynamic approaches indicates the need to trace certain critical requirements explicitly. For requirements that are not explicitly traced, information retrieval methods such as Poirot (our probabilistic IR tool), can be used to detect the impact of changes upon the goal tree so as to identify higher level goals that may have been adversely affected by the change. EBT style links are then used to re-evaluate any architectural assessment model linked to the goal. In this way GCT provides a more holistic approach to tracing NFR and supports both impact detection and analysis of NFRs and of the related architectural components.

4.2 An Example of Tracing to an AAM

Previously NFRs have been traced to AAMs in only limited ways. The following example previously reported in [5] and taken from the telephony domain, traces performance requirements to a system execution model. The execution model depicts resource contention resulting from multiple users trying to access the system at the same time. (Note that the numbers in this model assume steady state operation and do not take into account ringing or talking times.) Traceability links were established between the AAM and performance requirements. Certain variables in the model are categorized as input variables and represent parameters that are likely to change. For example one link was established between an assumption recorded in the requirements specification stating that "the average Media Gateway Controller (MGC) service time is 200 milliseconds" and the variable labeled *MST_MG*, representing the mean service time for originating and terminating Media Gateway Controller. Similar input links were created for the variable labeled *MST_MGC*, representing the mean service time for originating and terminating Media Gateways and for *SystemArrRate* (System arrival rate). An additional set of links were established between output variables of the model and specific goals within the requirements specification. Output variables include *AverageResponseTime* and *CallSetupResponseTime*.

These traceability links were then used to support impact analysis of a proposed change, which in this case was the introduction of a new Softswitch that would increase the average media gateway service time (MST_MG). EBT processed the impact analysis request through saving the current information in the system execution model, injecting the average MGC service time of the new softswitch into the impacted variable (*MSG_MG*), recalculating the CallSetUpResponseTime (ie re-executing the model), retrieving the results for reporting purposes, and then restoring the model to its initial state [4,5]. The results in the output variable *CallSetupResponseTime* were then compared against a performance requirement defining acceptable response time to determine if the proposed change would impact the goal.

In this example, traceability was established between a performance requirement and its appropriate AAM. However GCT extends this idea to trace a much broader spectrum of NFRs and incorporate a more diverse set of AAMs. Furthermore, GCT evaluates the tradeoffs and interdependencies that exist between NFRs and AAMs rather than just considering each one in isolation.

5. CONCLUSIONS

This position paper has discussed the critical role played by NFRs in the architectural design process and has explained why the interdependencies between NFRs and software architecture introduce special traceability challenges. Existing traceability techniques provide inadequate support for impact analysis of NFRs and this may be the reason that few organizations make any significant effort to trace NFRs. Simply detecting the impact of a change upon system-wide NFRs is not enough. The traceability mechanism should also provide support for evaluating the impact of the change within the architectural context. In GCT this is accomplished through establishing dynamically executable traces between NFR goals and AAMs.

Although the Goal Centric Traceability model is still in its formative stages, it outlines a method for improving the traceability of NFRs, by utilizing artifacts that are constructed during initial system development and making them available throughout the lifetime of the software system to support change management and to help ensure that critical architectural qualities are maintained over the long-term.

ACKNOWLEDGMENTS

The work on goal centric traceability is partially funded by NSF grant # 0447594. The work on dynamic trace retrieval described in this paper was funded under NSF grant # 0306303.

REFERENCES

[1] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering Traceability Links between Code and Documentation. IEEE Transactions on Software Engineering, Volume 28, No. 10, October 2002, 970-983.

[2] R.J.A. Buhr, "Use Case Maps as Architectural Entities for Complex Systems", *IEEE Transactions on Software Engineering*, Vol. 24, No. 12, Dec. 1998, pp. 1131-1151.

[3] R. Chitchyan, A. Rashid, P. Sawyer, A. Garcia, M. Pinto Alarcon, J. Bakker, B.Tekinerdogan, S. Clarke, A. Jackson, *Survey of Aspect Oriented Analysis and Design Approaches,* AOSD-Europe Network of Excellence, http://www.aosd-europe.net/documents/analys.pdf

[4] J. Cleland-Huang, C. K.Chang, H. Hu, K. Javvaji, G. Sethi, and J. Xia, "Requirements Driven Impact Analysis of System Performance", IEEE Proc. of the Joint Conference on Requirements Engineering, Essen, Germany, Sept. 2002.

[5] J. Cleland-Huang, C.K. Chang, and J. Wise, "Automating Performance Related Impact Analysis through Event Based Traceability", *Requirements Engineering Journal*, Springer-Verlag, Vol. 8, No. 3, Aug. 2003, pp. 171-182.

[6] J. Cleland-Huang and D. Schmelzer, "Dynamically Tracing Non-Functional Requirements through Design Pattern Invariants", Workshop on Traceability in Emerging Forms of Software Engineering, in conjunction with IEEE International Conference on Automated Software Engineering, October, 2003.

[7] Jane Cleland-Huang, Grant Zemont, and Wiktor Lukasik, "Heterogeneous Solutions for Improving the ROI of Requirements Traceability", *IEEE International Requirements Engineering Conference*, Kyoto, Japan, Sept. 8-10, 2004.

[8] Jane Cleland-Huang, Raffaella Settimi, Oussama BenKhadra, Eugenia Berezhan, Selvia Christina, "Goal Centric Traceability for Managing Non-Functional Requirements", *International Conference on Software Engineering*, St Louis, USA, May 2005.

[9] L. Cysneiros, J. Sampaio de Prado, Leite, "Nonfunctional Requirements: From Elicitation to Conceptual Models", *IEEE Transactions on Software Engineering*, Vol. 30, No. 5, May 2004. pp. 328-350

[10] O. Gotel, and A. Finkelstein, "An Analysis of the RequirementsTraceability Problem," Proc. First Int'l Conf. Requirements Eng., pp. 94-101, 1994.

[11] Daniel Gross and Eric Yu, "From Non-Functional Requirements to Design through Patterns", Requirements Engineering Journal, Vol 6, No. 1, 2001, pp. 18-36.

[12] Jane Huffman Hayes, Alexander Dekhtyar, Senthil Sundaram, Sarah Howard, "Helping Analysts Trace Requirements: An Objective Look," in Proceedings of IEEE Requirements Engineering Conference (RE) 2004, Kyoto, Japan, September 2004, pp. 249-261.

[13] R. Kazman et al., "Scenario-Based Analysis of Software Architecture," IEEE Software, Vol. 13, No. 6, Nov. 1996, pp. 47-55.

[14] R. Kazman, M. Klein, M. Barbacii, T. Longstaff, H.Lipson, and J. Carriere, "The Architecture Tradeoff Analysis Method", 4th IEEE International Conference on Engineering of Complex Computer Systems, Monterey, CA. 1998, pp. 68-78.

[15] A. Marcus, and J. Maletic. "Recovering Documentation-to-Source Code Traceability Links using Latent Semantic Indexing," Proceedings of the Twenty-Fifth International Conference on Software Engineering 2003, Portland, Oregon, 3 – 10 May 2003, pp. 125 – 135.

[16] B. Nuseibeh, "Weaving Together Requirements and Architecture", IEEE Computer, Vol. 34, No. 3, March 2001, pp. 115-117.

[17] J. D. Palmer, "Traceability", Software Requirements Engineering, R.H. Thayer and M. Dorfman (Eds). IEEE Computer Society Press, New York, 1997.

[18] D.L.Parnas, Studying Design Practice – Empirical Findings and Future Challenges, One-day seminar at the Dept. of Informatics, University of Trondheim, Oct. 29th, 1993.

[19] K. Pohl, R. Dömges, M. Jarke: Towards Method-Driven Trace Capture. CAiSE 1997: 103-116