Software Architecture Adaptability: An NFR Approach

Nary Subramanian
Applied Technology Division
Anritsu Company
Richardson, TX, USA
Phone No. 1-469-330-3333
narayanan.subramanian@anritsu.com

Lawrence Chung
Dept. of Computer Science
University of Texas, Dallas
Richardson, TX, USA
Phone No. 1-972-883-2178
chung@utdallas.edu

ABSTRACT
Adaptation of software systems is almost an inevitable process, due to the change in customer requirements, needs for faster development of new, or maintenance of existing, software systems, etc. No doubt numerous techniques have been developed to deal with adaptation of software systems. In this paper we present an overview of some of these techniques. As the first step in the development of software solution it is our opinion that software architecture should itself be adaptable for the final software system to be adaptable. In order to systematically support adaptation at the architectural level, this paper adapts the NFR (Non-Functional Requirements) Framework and treats software adaptability requirement as a goal to be achieved during development. Through this adaptation, then, consideration of design alternatives, analysis of tradeoffs and rationalization of design decisions are all carried out in relation to the stated goals, and captured in historical records. This NFR approach can also be adapted to a knowledge-based approach for (semi-)automatically generating architectures for adaptable software systems and we also discuss how this can be achieved.

Categories and Subject Descriptors
Software -Software Engineering - Requirements/Specifications (D.2.1): Methodologies (e.g., object-oriented, structured); Software -Software Engineering - Management (D.2.9): Software process models (e.g., CMM, ISO, PSP); Computing Methodologies -Simulation and Modeling - Model Development (I.6.5);

General Terms
Design

Keywords
Non-Functional Requirements, Adaptability, Software Architecture, NFR Framework, Knowledge Base

1. INTRODUCTION
Adaptation of software systems is almost an inevitable process. In fact it may even make sense to view adaptation as an important part of the software development lifecycle. The factors requiring software adaptation are several and include changing customer requirements, need for faster development of new software, adding new software features, and fixing software defects during the maintenance phase of software lifecycle. Since maintenance phase consumes about 50% of software development cost [1,48], an adaptable software system could perhaps save a large fraction of this cost.

There are several examples of software adaptation and some of these are mentioned below:

1. A dual-mode cell phone that automatically switches between the two systems depending upon currently available service.
2. Dynamic uploading of firmware without need to reboot the system.
3. A command-processing system that is capable of accepting commands of different versions.
4. A software system being able to operate on different OS - Solaris, Windows.
5. Performing system maintenance functions such as backup or garbage collection when the system is least busy.
6. A dynamically changeable format - from 2 digit year to 4 digit year; change units of measurement (the problem that caused the failure of Mars Climate Orbiter [49]).
7. Mars Pathfinder project [4] could be salvaged as the software had the ability to be modified in the field.
8. eLiza project at IBM [5] - develops self-managing systems.

Before proceeding further it may perhaps be worthwhile to define adaptation or adaptability. However, here we run into a problem of many unclear and inconsistent definitions in the literature. We give a representative sample below:

1. "Self-adaptive software modifies its own behavior in response to changes in its operating environment." [6].
2. "Adaptability is defined as the ease with which a system or parts of the system may be adapted to the changing requirements." [7].
3. "A program is called *adaptable* if it can be easily changed. A program is called *adaptive* if it changes its behavior automatically according to its context." [8].
4. "…a software quality metric that can be used to assess the ease with which software allows differing system constraints and user needs to be satisfied." [9].
5. "Attributes of software that bear on the opportunity for its adaptation to different specified environments without applying other actions or means than those provided for this purpose for the software considered." [10].

6. “The objective of adaptive maintenance is to evolve any system to meet the needs of the user and business.” [2].
7. “Adaptive Measures: a category of quality measures that address how easily a system can evolve or migrate.” [11].
8. “Adaptation is an organism’s or organization’s ability to alter its internal rules of operation in response to external stimuli.” [12].
9. “Over time, the original environment (e.g., CPU, operating system, business rules, external product characteristics) for which the software was developed is likely to change. Adaptive maintenance results in modification to the software to accommodate changes to its external environment.” [13].
10. “Adaptive evolution changes the software to run in a new environment.” [14].

Numerous techniques have been developed to deal with adaptation of software systems and each of these techniques has its own context of applicability. In this paper we present an overview of some of these techniques. However, very few of these techniques trace the solutions developed to their requirements.

In order to illustrate the approach that we have taken to tackle adaptability we give our definition of this NFR. Our definition is consistent with the spirit of [3]. An NFR such as adaptability tends to be a global property of a software system. In order to ensure that the software system finally developed exhibits the NFRs required, it is our opinion that the NFRs such as adaptability should be considered at the first step in the development of software solution, viz., in the software architecture itself. In order to systematically support adaptation at the architectural level, we adapted the NFR Framework [15,16,17,50]. The NFR Framework allows goal-oriented development of software and software adaptability is treated as one of the goals to be achieved during development. Through this adaptation, consideration of design alternatives, analysis of tradeoffs and rationalization of design decisions are all carried out in relation to the stated goals, and captured in historical records.

While the NFR Framework helps develop adaptable software systems, we adapted the NFR Framework to develop a knowledge-based system that will develop adaptable software architectures based on the requirements. We discuss this idea later in this paper. This knowledge base can also capture the services, as proposed in [2].

In this paper we have used the words adaptation and evolution somewhat synonymously. We first present in the Related Work section, a summary of some of the techniques used for dealing with adaptability; in the subsequent section, The NFR Approach, we present our definition of adaptability, introduce the NFR Framework and illustrate the use of the NFR Framework; in the next section, Knowledge-Based Approach, we demonstrate how the NFR Framework can be used to (semi-)automatically develop adaptable systems using a knowledge base of NFR Framework components; and in the final section we conclude our work.

2. RELATED WORK

We need a methodology consisting of notations, methods/techniques, and guidelines, that also allows for establishing traceability to the “whys” of the techniques, viz., the requirements. Our partial survey of the existing literature on adaptation has led us to categorize techniques used to deal with adaptation into the following: architecture-based techniques, component-based techniques, code-based techniques, genetic algorithm techniques, dynamic adaptation techniques, and adaptation methodologies.

A comprehensive adaptation technique that spans various adaptation requirements is given in [6]. In this technique, an adaptable system has embedded in it two managers – one for adaptation and the other for evolution. The adaptation manager takes high level decisions which are implemented by the evolution manager; the entire process is iterative. In [18] a framework for real-time software system adaptation, called RESAS (Real-time Software Adaptation System) is proposed. This framework permits a real-time system to adapt to timing constraints and to hardware failures. In [19] an adaptive software architecture (called multigraph architecture) for digital signal processing applications has been developed. This architecture, which is a signal flow graph, permits dynamic changing of graph nodes, to alter the execution sequence on the fly. In [20] the Odyssey architecture is proposed. Here, the operating system controls the fidelities of various applications running on a mobile phone based on the rate of power consumption, the aim being to conserve power as much as possible or as much as required by the user of the mobile. In [21] the Simplex Architecture has been described. This architecture permits online evolution of real-time systems by using a middleware that talks to the various components of the architecture using the publish/subscribe mechanism. This mechanism lets new components be created on the fly and replace existing components. In [22], the VEHICLES developing environment of NASA has been described. VEHICLES provides flexible developing environment which has been used for developing mission-critical systems.

Component-based techniques hope to leverage the advantages of component-based software development. In [23] domain-specific software architecture (DSSA) is used for evolution. Any architecture of a software system is an instance of the DSSA and evolution is achieved by creating another instance of the DSSA with the needed modifications. Another way to evolve is to use coordination contracts [24, 25] in an object-oriented environment. A coordination contract is a set of rules and constraints on the interaction between any two objects and the contract superposes its behavior on the interaction of the two objects between which the contract is valid. Adaptation is achieved by changing the contract and not the objects. Design components [26] are another way of adapting. Design components are collections of design patterns [27]. The design components can be customized and composed to form software architectures. Another way is to provide an adaptable interface to each component [28]. The code in the adaptable interface can be changed as required to achieve the needed adaptation without making any changes to the component. Yet another way to achieve adaptation is to use delegation [29] between objects in an object-oriented environment. In delegation the "this” parameter is set to the caller of the method and not to the callee of the method. This lets the caller call different objects to achieve the needed adaptation. Languages such as DARWIN, LAVA and JAVA support delegation. In [30], binary component adaptation is performed for changing Java classes on the fly. The byte code of any class that has to be adapted is changed just before the class is loaded into

the JVM. The application of fuzzy logic to component adaptation is described in [31]. Components are modeled using fuzzy membership functions which are then trained to adapt using different algorithms. Superimposition has been used for adaptation of components in [32]. In superimposition different behavior can be superimposed on an object to change its original behavior in a manner transparent to the object's clients. In [33] a real-time component factory for developing adaptable components for distributed systems is proposed. The real-time component factory develops components that can be customized for task priority and exception handling policy, by providing specific interfaces for customizing these services.

The code-based techniques change the software code to achieve adaptation. [34] is one of the first papers on adapting systems based on code. The paper proposes a scheme to develop adaptable systems. In [1], an extensible system called Extension Interpreter (EI) is proposed. EI works on an UNIX environment and permits new commands to be added to the system while the system is running. [35] discusses the problems faced and solutions found in the STARS (Software Technology for Adaptable, Reliable Systems) project for ARPA. A calculus for program adaptation can be found in [36]. This paper develops mathematical models for program adaptation based on incrementation, merging, modification and composition. Using these models any program adaptation along these methods can be mathematically achieved. [38] gives some "laws" on software evolution.

There are techniques that use genetic algorithms [31, 37] to deal with adaptation. Here the evolution of individual components occurs via the processes of recombination and mutation. Then the suitable components for the environment are selected.

There are techniques that adapt a software system dynamically, i.e., when the software system is running. Some of these techniques have been mentioned earlier [19,29,30]. In [39] a connector-based adaptation is described. The connectors, called co-operative action (CO action) are treated as first-class entities. Each CO action describes a collaboration between classes. In order to achieve adaptation, the collaborations between classes are changed on the fly, without changing the classes.

In [40] a methodology for designing adaptive applications is discussed. One of the points made is that the user should be involved in decisions to determine the extent to which an application should adapt in the given environment. In [41], machine learning has been recommended as a way to adapt general solutions. Using different learning algorithms, solutions for specific situations can be developed. In [42], a software evolution process has been described. In this process, firstly the requirements specifications are iteratively developed in consultation with the user. The design specification is then developed; however, during verification, each design specification should be a refinement of the corresponding requirement specification and not an evolution of the latter. Finally the implementation should be a refinement of the design and not an evolution. In [43], the EVO method used at HP is described, wherein several incremental cycles are used. [44] gives some requirements that an adaptable system should satisfy, viz., extensibility, flexibility, performance tunability and fixability. In [12] the adaptive software development methodology is proposed. This methodology has three steps: speculate (which gives the general idea of where to go in building the software system), collaborate (shared development of software) and learn (from experience).

3. THE NFR APPROACH

The various techniques and methodologies that we have mentioned in the previous section certainly have deepened our understanding of the nature of adaptability, especially in the particular domains considered and with the particular definitions given. The NFR Approach that we propose here is intended to be applicable to any such techniques or methodologies, regardless of the domain and definition of adaptability. Instead of proposing a single solution, the NFR Approach allows alternative solutions to be explored. Additionally, the NFR approach allows for decomposition of the NFR adaptability depending on the domain, or the application, and permits criticalities to be allocated to different NFRs of the decomposition. Also, the NFR approach allows for the consideration of design tradeoffs, as well as an interactive assessment of the degree to which NFRs such as adaptability are achieved.

3.1 Definition of Software Adaptability

The definition for software adaptability that we give below is consistent with the spirit of [3]. This definition has been mentioned earlier in [45, 46, 47], and re-presented below.

Adaptation means change in the system to accommodate change in its environment. More specifically, adaptation of a software system (*S*) is caused by change (δ_E) from an old environment (*E*) to a new environment (*E'*), and results in a new system (*S'*) that ideally meets the needs of its new environment (*E'*). Formally, adaptation can be viewed as a function:

Adaptation: $E \times E' \times S \rightarrow S'$, where *meet*(*S'*, *need*(*E'*)).

A system is adaptable if an adaptation function exists. Adaptability then refers to the ability of the system to make adaptation.

Adaptation involves three tasks:

1. ability to recognize δ_E
2. ability to determine the change δ_S to be made to the system S according to δ_E
3. ability to effect the change in order to generate the new system *S'*.

These can be written as functions in the following way:

EnvChangeRecognition : $E' - E \rightarrow \delta_E$

SysChangeRecognition : $\delta_E \times S \rightarrow \delta_S$

SysChange : $\delta_S \times S \rightarrow S'$, where *meet(S', need(E'))*.

The *meet* function above involves the two tasks of validation and verification, which confirm that the changed system (*S'*) indeed meets the needs of the changed environment (*E'*). The predicate *meet* is intended to take the notion of goal satisficing of the NFR Framework [15,16,17,50], which assumes that development decisions usually contribute only partially (or against) a particular goal, rarely "accomplishing" or "satisfying" goals in a clear-cut

sense. Consequently generated software is expected to satisfy NFRs within acceptable limits, rather than absolutely.

Figure 1explains the relationship between the various symbols described above.

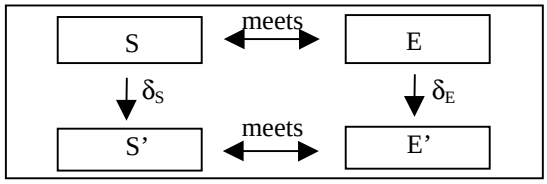

Figure 1. Symbols in the Definition of Adaptation

3.2 The NFR Framework

As per the NFR Framework [15,16,17,50], the following interleaving, iterative steps are carried out, through a visual representation called softgoal interdependency graph (hereafter, *SIG*), towards the generation of an adaptable software architecture:

1. Develop the NFR goals and their decomposition
2. Develop architectural alternatives
3. Develop design tradeoffs and rationale
4. Develop goal criticalities
5. Evaluation and Selection

In the subsequent sections, the above steps will be followed, starting with the development of the relevant NFR goals and decompositions. The architectural alternatives for the problem at hand will then be developed. Next the extent to which each of the architectures satisfies (or satisfices) the NFR goals will be determined based on the design tradeoffs and rationale; the criticalities of the goals will be determined at this stage or when the NFR goal decomposition is done. Finally the architectures will be evaluated. Selection of the most suitable architecture is assumed to follow in consideration of the characteristics of the intended application domain. Further details can be seen in [15, 16, 17, 50].

+ +	Strongly Positive Satisficing or MAKES
+	Positive Satisficing or HELPS
-	Negative Satisficing or HURTS
--	Strongly Negative Satisficing or BREAKS

Figure 2. Codes for Satisficing Degrees

Figure 2 gives the codes used in a SIG, while Figures 3 and 5 gives two sample SIGs. Other notations used in a SIG are explained here. The light cloud indicates an NFR softgoal, i.e., it is an NFR. The dark cloud indicates a design softgoal, i.e., it is a design representation, an architecture or a component of an architecture. The lines between the dark clouds and the light clouds indicate the degree to which the design component corresponding to the dark cloud satisfices the NFR represented by a light cloud. If there is no line between them it means that the design component does not satisfice the NFR in any manner,

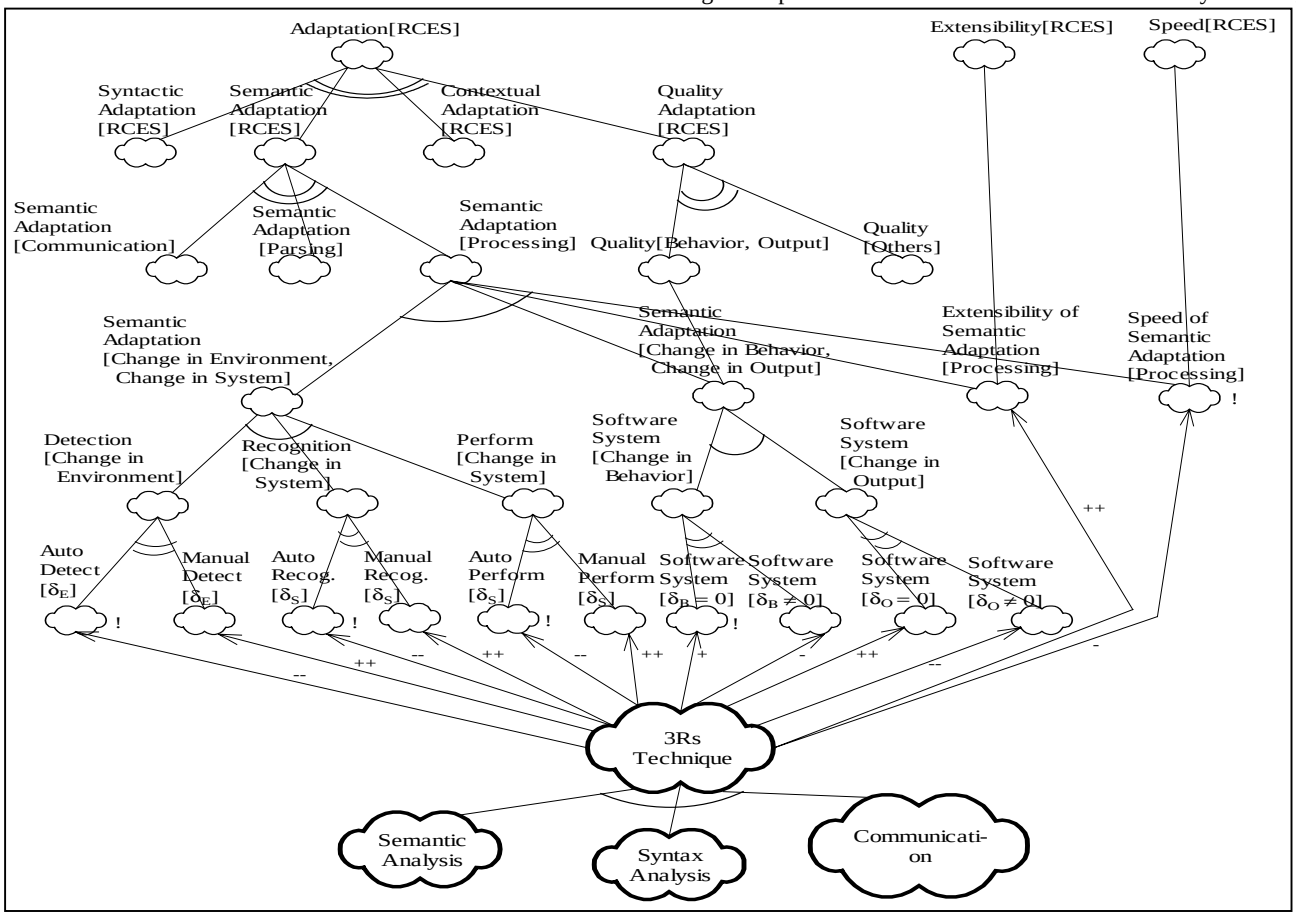

Figure 3. Example SIG -1

either positively or negatively. Satisficing can occur in four intensities (as indicated by the codes of Figure 2) – strongly positively satisficing, positively satisficing, negatively satisficing and strongly negatively satisficing. Determination of the degree of satisficing takes place during step 3 above. The goal criticalities are indicated by '!' next to the softgoals – a single '!' indicates that the softgoal is critical while '!!' indicates that the softgoal is severely critical. The goal criticalities are determined during step 4 above. In addition, a single arc means an AND of all the softgoals originating from a softgoal, while a double arc means an OR of all the sub-softgoals of a softgoal. Finally, in the evaluation step, an interactive labeling procedure is used to label the leaf NFR softgoals based on how much they are satisficed and their criticalities, and propagate the labels up the SIG, while taking into consideration the various contribution types.

In Figure 3, an example SIG is given (taken from [47]). The clouds with dark borders are design softgoals while normal clouds are NFR softgoals. The NFR softgoals have the following nomenclature:

Type[Topic1, Topic2, …],

where *Type* is a non-functional aspect (e.g., adaptability) and *Topic* is a system to which the *Type* applies (e.g., RCES, which stands for Remotely Controlled Embedded System), and the refinement can take place along the *Type* or the *Topic*. The high-level NFR softgoals (at the top of the figure) are decomposed as required for the application domain into constituent NFR softgoals. The design softgoals represent the design components for a particular solution. The links between the design softgoals and the NFR softgoals indicate the contribution that the design softgoals make for the various NFR softgoals. There are reasons for the different types of the links, and they can be shown by the third type of softgoal in the NFR Framework, viz., claim softgoals. The links between the NFR softgoals themselves or that between the design softgoals themselves are referred to as interdependencies.

Figure 5 gives yet another example of SIG (taken from [46]). In this figure, RCA stands for a system called the Radio Communication Analyzer, while GPIB stands for the General Purpose Interface Bus. Here the claim softgoals are shown as dotted clouds. The curly double-headed arrow indicates for satisficing which NFR softgoal the corresponding design softgoal was developed. The values inside the NFR softgoals are labels which represent the metric for the corresponding softgoal.

Due to space restrictions, we are unable to describe in greater detail the systems pertaining to the SIGs of Figures 3 and 5; however, the details can be found in the references [46, 47]. More examples can be found in [15].

3.3 The NFR Approach to Adaptability

In applying the NFR Framework to adaptability, based on the definition of adaptability and the domain requirements, the NFR softgoal adaptability is first decomposed into its constituent NFRs – this is illustrated by the top part of the SIGs in Figures 3 and 5. Then a determination of the extent of satisficing of the various NFR softgoals by the design softgoals (for a particular architecture) is made. This is represented by the satisficing links between the upper part and the lower part of the SIGs in Figures 3 and 5. Whether the design softgoals meet the requirements can be decided by the developer from the SIG.

Another way to use the NFR Framework is in the knowledge base approach discussed next. Advantages of the NFR Framework are several including:

1. the history of design decisions is recorded in graphs
2. development knowledge can be organized into catalogs
3. it is very easy to recollect past decisions from the graphs.

4. KNOWLEDGE-BASED APPROACH

The goal of the knowledge-based approach is to develop a knowledge base of NFR (adaptability) decomposition methods, various design softgoals, the extent to which the design goals satisfice the NFR softgoals, and the like. Then based on a chosen decomposition of NFR adaptability it will be possible to automatically or semi-automatically generate an architecture that will satisfice the various NFR softgoals. This knowledge base can also capture the services, as proposed in [2].

In the knowledge-based approach catalogs of the various components of the NFR Framework and their inference rules are developed. Then based on an initial set of non-functional requirements and an initial set of functional requirements, the various catalogs are searched to develop the SIG. Using the SIG, the final architecture can be developed either automatically or semi-automatically (by the developer using the SIG).

How catalogs can be developed for adaptability NFR will be described in this section. Frame-like notations will be used to describe the various components of the NFR Framework.

4.1 Capturing Softgoals

Firstly, each of the three types of softgoals, viz., NFR softgoal, design softgoal and claim softgoal, can be represented in frame-like notation. This is shown for the NFR softgoal Adaptation[RCES] of Figure 3 at the top of Figure 4 and the softgoal is reproduced at the bottom of Figure 4.

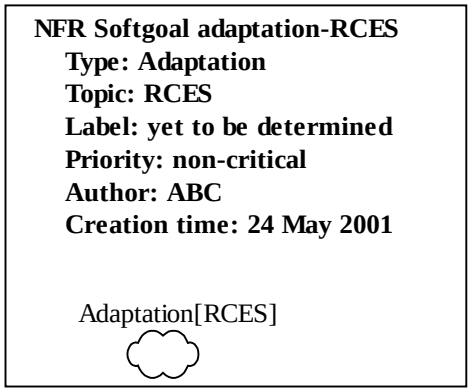

Figure 4. Representation for NFR Softgoal

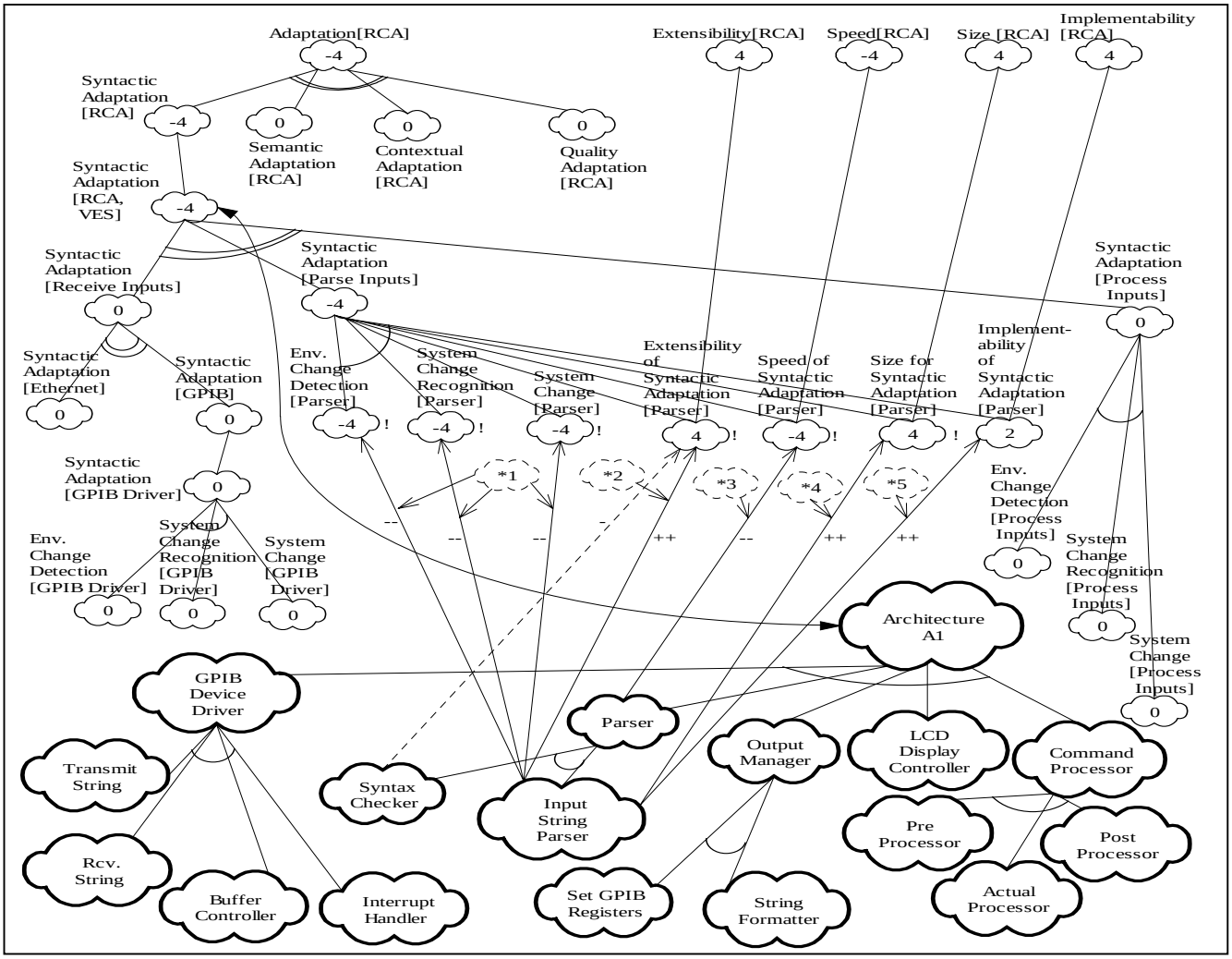

Figure 5. Example SIG-2

For the design softgoal (formally called as operationalizing softgoal in the NFR Framework) String Formatter of Figure 5, the frame-like notation will be as given in the top of Figure 6, with the corresponding softgoal reproduced at the bottom of Figure 6.

Operationalizing Softgoal stringformatter
Type: String Formatter
Topic: Output String
Label: yet to be determined
Priority: non-critical
Author: ABC
Creation time: 24 May 2001

String Formatter

Figure 6. Representation of an Operationalizing Softgoal

For the claim softgoal marked *1 in Figure 5, the frame-like notation will be as given in Figure 7. Based on these frame-like notations, the knowledge base could be populated with various softgoals.

Claim Softgoal claim1
Type: claim1
Topic: *1 /*could be for example,
"Based on Domain Knowledge"*/
Label: yet to be determined
Priority: non-critical
Author: ABC
Creation time: 24 May 2001

*1

Figure 7. Representation of a Claim Softgoal

4.2 Refinement Methods

Based on the knowledge about softgoals, methods for refining (decomposing) softgoals could be developed and put in the knowledge base.

4.2.1 NFR Softgoal Decomposition Methods

The decomposition of the NFR softgoal Adaptation[RCES] in Figure 3 may be governed by the method given on top of Figure 8, and the decomposition is repeated on the bottom of Figure 8.

Given, the NFR Type catalog as:

Subtypes of Adaptation are Syntactic Adaptation, Semantic Adaptation, Contextual Adaptation and Quality Adaptation

then the decomposition method of Figure 8 that uses the above type catalog as an inference rule results in the four offsprings of the NFR softgoal Adaptation[RCES] shown in Figure 8. An NFR DecompositionMethod RCAQualityViaSubType defined similar to the method RCESQualityViaSubType, where the topic RCES is replaced by RCA, will result in the decomposition of the NFR softgoal Adaptation[RCA] shown in Figure 5. The decomposition of Figure 8 can be represented in a short form as:

Syntactic Adaptation[RCES] OR Semantic Adaptation[RCES] OR Contextual Adaptation[RCES] OR Quality Adaptation[RCES] SATISFICE Adaptation[RCES]

Another NFR softgoal decomposition method may be via subclass. Thus if in the functional requirement, the class hierarchy is :

Receive Inputs is the superclass of Ethernet class and GPIB class

then the decomposition method at the top of Figure 9 will result in the decomposition of the NFR softgoal Syntactic Adaptation[Receive Inputs] shown in Figure 5, repeated again for convenience, at the bottom of Figure 9. In short form, this decomposition can be written as:

Syntactic Adaptation[Ethernet] OR Syntactic Adaptation[GPIB] SATISFICE Syntactic Adaptation [Receive Inputs]

Using such NFR decomposition methods, the NFR Adaptation was decomposed into its sub-softgoals as shown in Figures 3 and 5.

4.2.2 Operationalization Methods

After the decomposition of the NFR softgoal, operationalizing methods are used to develop the design softgoal hierarchy part (the lower part of Figures 3 and 5) of the SIG. Operationalizing methods refine NFR softgoals into operationalizing softgoals or operationalizing softgoals into further, more specific operationalizing softgoals. Operationalizations represent possible design or implementation components.

An example of an operationalization method is given in Figure 10, at the top. When this method is applied to the NFR softgoal

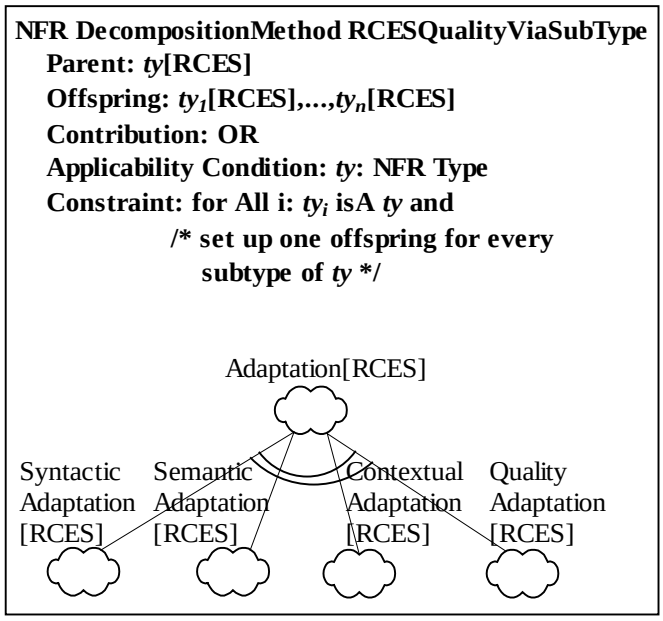

Figure 8. Representation of an NFR Decomposition Method

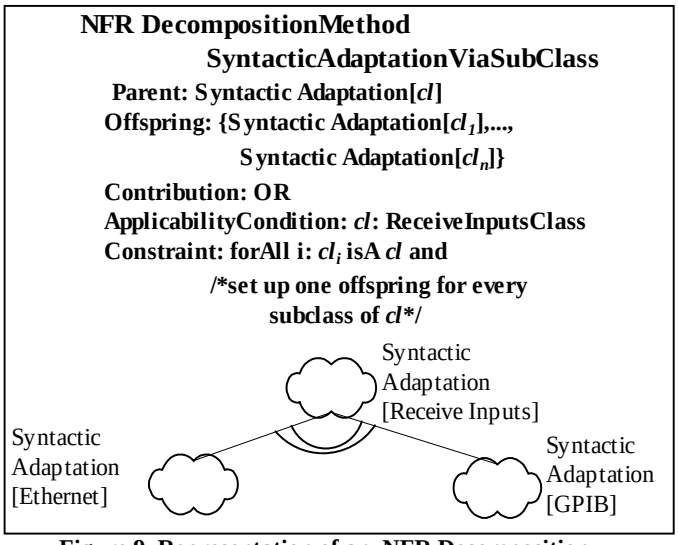

Figure 9. Representation of an NFR Decomposition Method based on Sub Class

Speed of Syntactic Adaptation[Parser] of Figure 5, the result is shown in Figure 5 and is repeated at the bottom of Figure 10, with the -- symbol next to the arrow signifying the BREAK contribution. Likewise, an operationalizing method, say *3RsTechniqueViaSubType*, could be defined to decompose the design softgoal named 3RsTechnique in Figure 3 into its offspring softgoals.

Using such operationalization methods the rest of the SIG of Figures 3 and 5 can be completed.

4.2.3 Argumentation Templates

One additional item in the SIG of Figure 5 is the claim softgoals. They are supported by argumentation templates in the NFR

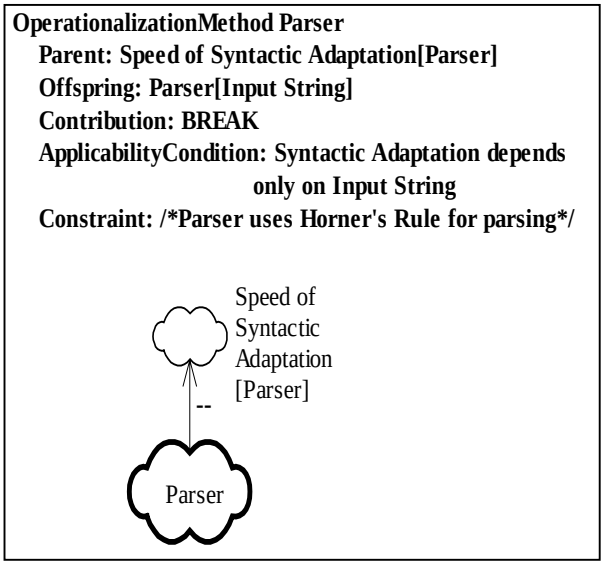

Figure 10. Representation of an Operationalization Method Applied to an NFR Softgoal

Framework. Thus for the claim softgoal marked *2 in Figure 5, the template is shown on the top of Figure 11, with the relevant portion of Figure 5 repeated at the bottom of Figure 11, for convenience.

Rules can also be developed for the prioritization of some softgoals (indicated by '!' in Figures 3 and 5).

4.3 Correlation Rules

Rules can also be developed for correlations between NFR softgoals in the SIG. The interactions between NFR softgoals may be synergistic or conflicting. During the process of software development, interactions between softgoals are discovered and noted in a softgoal interdependency graph. Such discoveries can be made with the help of correlation rules. Thus in Figure 5, where the design softgoal Syntax Checker contributes negatively to the NFR softgoal Extensibility of Syntactic Adaptation[Parser] (shown by the dashed arrow with – symbol next to it), the correlation rule of Figure 12 could be used to indicate this fact with the corresponding SIG representation given at the bottom of Figure 12.

4.4 Using the SIG

Using the above methods, templates and rules, in the knowledge base, and based on the given initial non-functional requirements and functional requirements, the SIG can be developed automatically or with the assistance of the developer. Having developed the SIG, labels can be associated with leaf softgoals and these labels can be propagated from the bottom of the SIG to the NFR softgoals at the top of the SIG using a propagation scheme. This scheme may again be automatic algorithm or a semi-automatic algorithm. Using the scheme, the extent to which the (semi-)automatically generated architecture satisfices the various softgoals can be determined and the appropriateness of the architecture for the particular situation can be evaluated.

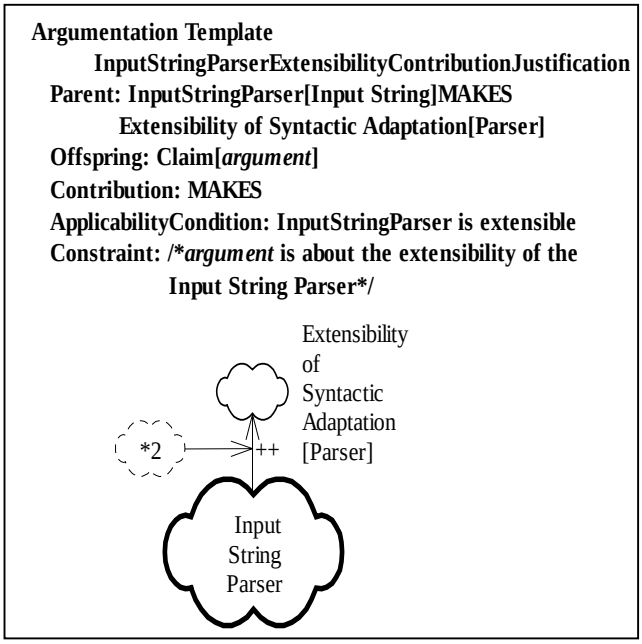

Figure 11. Representation of an Argumentation Template

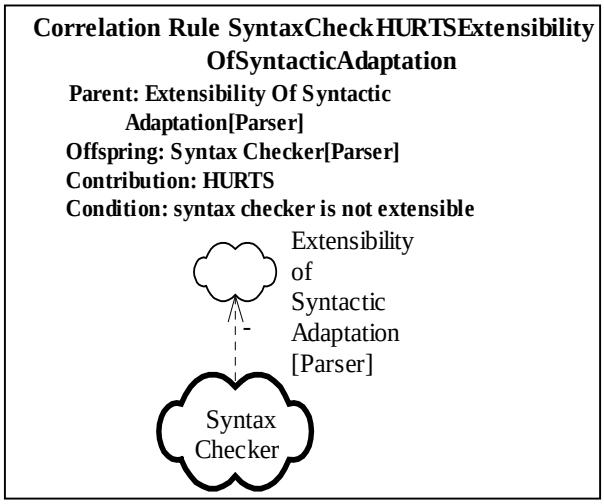

Figure 12. Representation of a Correlation Rule

5. CONCLUSION

In this paper we have discussed a way to deal with the important non-functional requirement (NFR) of software adaptation, along with some motivations for, and survey of, techniques for adaptation. We have also shown how to handle definitions of this NFR in the literature, which can often be unclear and even inconsistent, by taking an NFR Approach. This approach adapts the NFR Framework [15,16,17,50] which helps to systematically handle NFRs such as adaptability. We then illustrated how the NFR Approach could be extended into a knowledge-based approach in order to use and reuse the various techniques for achieving software architecture adaptability, with a discussion of how the knowledge-based approach would help to (semi-) automatically generate adaptable architectures.

The NFR Approach is consistent with the spirit of QFD [51] and Dr. Barry Boehm's Spiral Model/Win-Win system[52] – the NFR Approach provides enough constructs to easily extend these techniques. To date, the NFR Approach has been used in analyzing adaptation in some systems [45, 46, 47]. However, there still is a lot of work to be done. One concern is to find better cataloging of this NFR and those of its refinements. Another is to develop methods for different application domains so that the knowledge base better represents the needs of the industry. This will also allow industry practitioners to make use of the knowledge base for their own work. Also, right now we are not fully sure exactly how to generate the architectures. However, we believe that the NFR Approach to handling adaptability has already shown some signs of practical effectiveness and hopefully will be used by software practitioners.

6. ACKNOWLEDGEMENTS

We wish to thank the anonymous referees for their valuable suggestions and comments.

7. REFERENCES

[1] Notkin, D., Griswold, W.G. "Extension and Software Development", *Proceedings of the 10th International Conference on Software Engineering*, April 1988, pp. 274-283.

[2] Mikkonen, T., Lahde, E., Niemi, J., Siiskonen, M. "Managing Software Evolution with the Service Concept", *Proceedings of the International Symposium on Principles of Software Evolution*, Nov. 2000, Japan, IEEE Computer Press, pp. 46 – 50.

[3] Lehman, M.M. and Ramil, J.F. "Towards a Theory of Software Evolution – And its Practical Impact", *Proceedings of the International Symposium on Principles of Software Evolution*, Nov. 2000, Japan, IEEE Computer Press, pp. 2-11.

[4] http://java.sun.com/people/jag/pathfinder.html.

[5] http: //www-1.ibm.com/servers/eserver/introducing/eliza.

[6] Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S., and Wolf, A.L. "An Architecture-Based Approach to Self-Adaptive Software", *IEEE Intelligent Systems*, May/June 1999, pp. 54-62.

[7] Adaptability in Object-Oriented Software Development Workshop Report, 10th European Conference on Object-Oriented Programming, July 8-12, 1996, Linz, Austria.

[8] Workshop on Adaptable and Adaptive Software Report, Addendum to the Proceedings of the 10th Annual Conference on Object-Oriented Programming Systems, Languages and Applications, Oct. 15-19, 1995, Austin, TX, USA.

[9] Dorfman, M., and Thayer, R.H. (editors). *Standards, Guidelines, and Examples on System and Software Requirements Engineering*, IEEE Computer Society Press, Los Alamitos, California, 1990.

[10] Sanders, J., Curran, E. *Software Quality – A Framework for Success in Software Development and Support, Addison-Wesley*, Wokingham, England, 1994.

[11] Software Engineering Institute's website: http:\\www.sei.cmu.edu.

[12] Highsmith, J.A. *Adaptive Software Development – A Collaborative Approach to Managing Complex Systems*, Dorset House Publishing, New York, 1999.

[13] Pressman, R.S. *Software Engineering – A Practitioner's Approach*, 4th Edition, McGraw-Hill Companies, Inc., New York, 1997.

[14] Oreizy, P., Medvidovic, N. and Taylor, R.N. "Architecture-Based Runtime Software Evolution", Proceedings of the International Conference on Software Engineering", Kyoto, Japan, April 1998, pp. 177 - 186.

[15] Chung, L., Nixon, B.A., Yu, E. and Mylopoulos, J. *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, Boston, 2000.

[16] Mylopoulos, J., Chung, L., Nixon, B. "Representing and Using Nonfunctional Requirements: A Process-Oriented Approach", *IEEE Transactions on Software Engineering*, Vol. 18, No. 6, June 1992, pp. 483-497.

[17] Mylopoulos, J., Chung, L., Liao, S.S.Y., Wang, H., Yu, E. "Exploring Alternatives During Requirements Analysis", *IEEE Software*, Jan/Feb. 2001, pp. 2 – 6.

[18] Bihari, T. E. and Schwan, K. "Dynamic Adaptation of Real-Time Systems", *ACM Transactions on Computer Systems*, Vol. 9, No. 2, May 1996, Pages 143-174.

[19] Sztipanovits, J., Karsai, G., Bapty, T. "Self-Adaptive Software for Signal Processing", *Communications of the ACM*, Vol. 41, No. 5, May 1998, pp. 66-73.

[20] Flinn, J., Satyanarayanan, M. "Energy-aware Adaptation for Mobile Applications", *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, December 12-15, 1999, Charleston, USA, 48-63.

[21] Sha, L., Rajkumar, R., Gagliardi, M. "Evolving Dependable Real-Time Systems", *Proceedings of Aerospace Applications Conference*, Feb. 1996, pp. 335-346.

[22] Bellman, K.L. "An Approach to Integrating and Creating Flexible Software Environments Supporting the Design of Complex Systems", *Proceedings of the 1991 Winter Simulation Conference*, December 8 – 11, 1991, Phoenix, AZ, USA, pp. 1101 – 1105.

[23] Jarzabek, S., Hitz, M. "Business-Oriented Component-Based Software Development and Evolution", *Proceedings of the International Workshop on Large-Scale Software Composition*, August 28, 1998, Vienna, Austria, pp. 784-788.

[24] Koutsoukos, G., Goweia, J., Andrade, L., Fiadeiro, J.L. "Managing Evolution in Telecommunication Systems", from the website of Dr. Fiadeiro.

[25] Andrade, L.F., Fiadeiro, J.L. "Coordination: the Evolutionary Dimension", from the website of Dr. J. L. Fiadeiro.

[26] Keller, R.K., Schauer, R. "Design Components: Towards Software Composition at the Design Level", *Proceedings of International Conference on Software Engineering*, April 19-25, 1998, Kyoto, Japan, pp. 302-311.

[27] Gamma, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*, Reading, Massachusetts, Addison-Wesley, 1995.

[28] Heineman, G.T. "Adaptation and Software Architecture", *Proceedings of the 3rd International Workshop on Software Architecture*, Nov. 1-5, 1998, Orlando, Florida, USA, pp. 61-64.

[29] Kniesel, G. "Type-Safe Delegation for Run-Time Component Adaptation", *Lecture Notes in Computer Science 1628*, Springer-Verlag, Berlin Heidelberg, 1999, pp. 351-366.

[30] Keller, R., Holzle, U. "Binary Component Adaptation", *Lecture Notes in Computer Science 1445*, Springer-Verlag, Berlin Heidelberg, 1998, pp. 307-329.

[31] Chen, J., Rine, D.C. "Training Fuzzy Logic Based Software Components by Combining Adaptation Algorithms", *Soft Computing*, Vol. 2, Issue 2, pp. 48 – 60.

[32] Bosch, J. "Superimposition: A Component Adaptation Technique", *Information and Software Technology*, Volume 41, Issue 5, March 1999, pp. 257 – 273.

[33] Yau, S.S. and Karim, F. "Component Customization for Object-Oriented Distributed Real-time Software Development", *Proceedings of the 3rd IEEE International Symposium on Object-Oriented Real-time Distributed Computing*, March 15-17, 2000, pp. 156 – 163.

[34] Parnas, D.L. "Designing Software for Ease of Extension and Contraction", *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 2, March 1979, pp. 128-137.

[35] Davis, M.J. "Adaptable, Reusable Code", *Proceedings of the 17th International Conference on Software Engineering on Symposium on Software Reusability*, April 19-30, 1995, Seattle, WA, USA, pp. 38 – 46.

[36] Ayed, R.B., Desharnais, J., Frappier, M., Mili, A. "A Calculus of Program Adaptation and its Applications", *Science of Computer Programming*, Vol. 38, Issues 1 – 3, August 2000, 73 – 123.

[37] Spears, W.M., DeJong, K.A., Back, T., Fogel, D.B., de Garis, H. "An Overview on Evolutionary Computation", *Proceedings of European Conference on Machine Learning*, Vienna, Austria, April 1993, Lecture Notes in Artificial Intelligence 667, Springer-Verlag, Berlin Heidelberg 1993, pp. 442 – 459.

[38] Lehman, M.M., and Belady, L. *Program Evolution: Processes of Software Change*, Ch. 27, Acad. Press, London, 1985.

[39] de Lemos, R. "A Co-operative Object-Oriented Architecture for Adaptive Systems", *Proceedings of the 7th IEEE International Conference and Workshop on Engineering of Computer Based Systems*, April 2000, pp. 120-128.

[40] McIlhagga, M., Light, A., and Wakeman, I. "Towards a Design Methodology for Adaptive Applications", *The 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, October 25-30, 1998, Dallas, TX, USA, pp. 133-144.

[41] Gratch, J., DeJong, G. "A Statistical Approach to Adaptive Problem Solving", *Artificial Intelligence*, Vol. 88, Issues 1-2, December 1996, pp. 101-142.

[42] Liu, S. "Evolution: A More Practical Approach than Refinement for Software Development", *Proceedings of the 3rd IEEE International Conference on Engineering of Complex Computer Systems*, IEEE Computer Society Press, Villa Olmo, Como, Italy, September 8 –12, 1997, pp. 142-151.

[43] May, E.L., and Zimmer, B.A. "The Evolutionary Development Model for Software", *Hewlett-Packard Journal*, August 1996, pp. 39-45.

[44] Fayad, M., and Cline, M.P. "Aspects of Software Adaptability", *Communications of the ACM*, Volume 39, No. 10, Oct. 1996, pp. 58-59.

[45] Subramanian, N., and Chung, L. "Architecture-Driven Embedded Systems Adaptation for Supporting Vocabulary Evolution", *Proceedings of the International Symposium on Principles of Software Evolution*, IEEE Computer Press, Nov. 2000, pp. 144 – 153.

[46] Chung, L., and Subramanian, N. "Process-Oriented Metrics for Software Architecture Adaptability", to appear in the Proceedings of ISRE, 2001.

[47] Chung, L., and Subramanian, N. "Architecture-Based Semantic Adaptation: A Study of Remotely Controlled Embedded Systems", to appear in the Proc. of ICSM, 2001.

[48] Arthur, L.J. *Software Evolution – The Software Maintenance Challenge*, John Wiley & Sons, New York, 1988.

[49] http://mpfwww.jpl.nasa.gov/msp98/news/mco991110.html.

[50] Chung, L., and Nixon, B.A."Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach"; *Proc., IEEE 17th International Conference on Software Engineering*, Seattle, April 24-28, 1995., pp. 25-37.

[51] Hauser, J.R., and Clausing, D. "The House of Quality," *Harvard Business Review*, May--June 1988, pp. 63--73.

[52] Boehm, B., and In, H. "Aids for Identifying Conflicts Among Quality Requirements", Proceedings of ICRE '96, Colorado, April 1996.