Journal of Advances in Science and Technology
VOL. 3, NO.4, February-2012, ISSN 2230-9659

Software Architecture Adaptability: A Non-Functional Requirements Approach

Rajendra Singh

Research Scholar, Singhania University, Rajasthan, India

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);

----------♦----------

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:

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

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:

$$EnvChangeRecognition : E' - E \rightarrow \delta_E$$
$$SysChangeRecognition : \delta_E \times S \rightarrow \delta_S$$
$$SysChange : \delta_S \times S \rightarrow S', \text{ where } meet(S', need(E')).$$

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 1 explains the relationship between the various symbols described above.

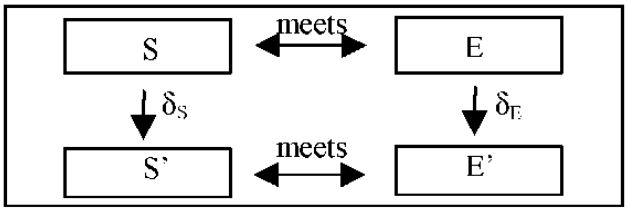

Figure 1. Symbols in the Definition of Adaptation

3.2 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:

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

4. 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.

REFERENCES

- Notkin, D., Griswold, W.G. "Extension and Software Development", *Proceedings of the 10th International Conference on Software Engineering,* April 1988, pp. 274-283.
- 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.
- 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. http://java.sun.com/people/jag/pathfinder.html. http: //www-1.ibm.com/servers/eserver/introducing/eliza.
- 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.

- Adaptability in Object-Oriented Software Development Workshop Report, 10th European Conference on Object-Oriented Programming, July 8-12, 1996, Linz, Austria.
- 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.
- 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.
- Sanders, J., Curran, E. *Software Quality - A Framework for Success in Software Development and Support, Addison-Wesley,* Wokingham, England, 1994.
- Software Engineering Institute's website: http:\\www. sei.cmu.edu.
- Highsmith, J.A. *Adaptive Software Development - A Collaborative Approach to Managing Complex Systems,* Dorset House Publishing, New York, 1999.
- Pressman, R.S. *Software Engineering - A Practitioner's Approach,* 4th Edition, McGraw-Hill Companies, Inc., New York, 1997.
- 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.
- Chung, L., Nixon, B.A., Yu, E. and Mylopoulos, J. *NonFunctional Requirements in Software Engineering,* Kluwer Academic Publishers, Boston, 2000.
- 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.
- 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.
- Bihari, T. E. and Schwan, K. "Dynamic Adaptation of RealTime Systems", *ACM Transactions on Computer Systems,* Vol. 9, No. 2, May 1996, Pages 143-174.
- 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.
- 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.
- Sha, L., Rajkumar, R., Gagliardi, M. "Evolving Dependable Real-Time Systems", *Proceedings of Aerospace Applications Conference,* Feb. 1996, pp. 335-346.
- 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.
- 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.
- Koutsoukos, G., Goweia, J., Andrade, L., Fiadeiro, J.L. "Managing Evolution in Telecommunication Systems", from the website of Dr. Fiadeiro.
- Andrade, L.F., Fiadeiro, J.L. "Coordination: the Evolutionary Dimension", from the website of Dr. J. L. Fiadeiro.
- 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.

- Gamma, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software,* Reading, Massachusetts, Addison- Wesley, 1995.
- 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.
- Kniesel, G. "Type-Safe Delegation for Run-Time Component Adaptation", *Lecture Notes in Computer Science 1628,* Springer-Verlag, Berlin Heidelberg, 1999, pp. 351-366.
- Keller, R., Holzle, U. "Binary Component Adaptation", *Lecture Notes in Computer Science 1445,* Springer-Verlag, Berlin Heidelberg, 1998, pp. 307-329.
- Chen, J., Rine, D.C. "Training Fuzzy Logic Based Software Components by Combining Adaptation Algorithms", *Soft Computing,* Vol. 2, Issue 2, pp. 48 - 60.
- Bosch, J. "Superimposition: A Component Adaptation Technique", *Information and Software Technology,* Volume 41, Issue 5, March 1999, pp. 257 - 273.
- 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.
- 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.
- 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.
- 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.
- 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.
- Lehman, M.M., and Belady, L. *Program Evolution: Processes of Software Change,* Ch. 27, Acad. Press, London, 1985.
- 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.
- 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.
- Gratch, J., DeJong, G. "A Statistical Approach to Adaptive Problem Solving", *Artificial Intelligence,* Vol. 88, Issues 1-2, December 1996, pp. 101-142.
- 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.
- May, E.L., and Zimmer, B.A. "The Evolutionary Development Model for Software", *Hewlett-Packard Journal,* August 1996, pp. 39-45.
- Fayad, M., and Cline, M.P. "Aspects of Software Adaptability", *Communications of the ACM,* Volume 39, No. 10, Oct. 1996, pp. 58-59.
- 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.
- Chung, L., and Subramanian, N. "Process-Oriented Metrics for Software Architecture Adaptability", to appear in the Proceedings of ISRE, 2001.

- Chung, L., and Subramanian, N. "Architecture-Based Semantic Adaptation: A Study of Remotely Controlled Embedded Systems", to appear in the Proc. of ICSM, 2001.
- Arthur, L.J. *Software Evolution - The Software Maintenance Challenge,* John Wiley & Sons, New York, 1988. http://mpfwww.jpl.nasa.gov/msp98/news/mco991110.html.
- 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.
- Hauser, J.R., and Clausing, D. "The House of Quality," *Harvard Business Review,* May--June 1988, pp. 63--73.
- Boehm, B., and In, H. "Aids for Identifying Conflicts Among Quality Requirements", Proceedings of ICRE '96, Colorado, April 1996.