

Metrics for Software Adaptability

Nary Subramanian¹, Lawrence Chung²

¹Applied Technology Division
Anritsu Company
Richardson, TX, USA
narayanan.subramanian@anritsu.com

²Department of Computer Science
University of Texas, Dallas
Richardson, TX, USA
chung@utdallas.edu

Abstract

This paper discusses our initial work in developing metrics for software adaptability. In this paper we have developed several metrics for software adaptability. One of the advantages of the metrics that we have developed is that they are applicable at the architectural level. Since architecture development is the first stage of the design process, the extent to which the architecture is adaptable will determine the adaptability of the final software. Hence the metrics in this paper will help determine the extent to which the final software will be adaptable as well.

This paper first defines software adaptability and then defines adaptability indices – the architecture adaptability index (AAI) and the software adaptability index (SAI). The application of these indices is illustrated with examples. Also the validity of these indices for hierarchical architectures, legacy systems and for dynamic adaptation is examined.

1 Introduction

Objective measurement of non-functional requirements (*NFRs*) in any software is one of the most difficult activities. This is because the inherent nature of the *NFRs* makes their common understanding difficult. The problem is compounded by the fact that the requirements for any software are usually vague about the *NFRs* that the software should satisfy and how to evaluate the *NFRs* in the final software. An ideal solution to these problems will be the development and usage of metrics for all *NFRs* – this will let the software be checked for the *NFRs* at all stages of its lifecycle and in case of deviation from the requirements corrective action may be taken.

While there are several *NFRs* such as performance, maintainability, reusability, security, and so on, among the more important of the *NFRs* is adaptability [2]. Intuitive definition of adaptability is the extent to which a software system adapts

to change in its environment. An adaptable software system can tolerate changes in its environment without external intervention. For example, a dual-mode cell phone can find out by itself if any one of the two wireless standards it supports is available at its current location and if so it starts using that standard. A practical metric for this NFR will help software developers and their organizations.

Usually good software is robust – it can tolerate some deviations in the environment. For example, if user presses character keys while entering numeric data the software can be designed to ignore such incorrect key presses. While robustness to the software can be added at the design or even the implementation stage, adaptability requirements cannot be added at such late stages. Adaptability differs from robustness in the scale of environment change – adaptable software can tolerate much larger deviations in the environment than a robust one. Adaptability can be enforced only if it is considered at the architecture development stage.

Several measures for adaptability have been proposed – Fenton [3] proposes a scale in terms of time spent in maintaining (the measure for maintainability, assuming adaptive maintenance), while Gilb [4] proposes some measures for extendability such as number of additions to an existing system. While these measures are certainly useful, it is the opinion of the authors that measures for software architecture are extremely important. This is because the quality of the final software product depends on the first stage of the solution – namely the software architecture [6,7]. Hence the measures developed at the software architectural level will very strongly predict the corresponding measures of the final software product. These ideas are also reflected in a paper by Duenas et al. [5] which defines adaptability (using the metric for changeability measure) as a parameterisation ratio. In this paper we propose a different technique for measuring adaptability in a software system – this technique is not restrictive and is flexible to suit the needs of different organizations in addition to being easy to understand and use. Besides the techniques we present here have the following properties:

1. they are evolvable – as the product development proceeds through its lifecycle the metrics can be updated continuously to find the latest value of adaptability in software
2. they are scalable – the metrics proposed in this paper can be tailored to suit the particular organization, the particular project or the particular software product.

In this paper the word metric is used to indicate a scale of measurement. This paper represents the initial work we did to measure the NFR adaptability. One of the problems in measuring adaptability is the definition of adaptability. In this paper we have given an intuitive definition of adaptability. We hope that organizations wishing to use this metric will use their own definition of adaptability.

In Section 2, we develop the ideas behind our metrics for software adaptability, in Section 3 we propose our adaptability metrics and illustrate their use with

examples, in Section 4 we discuss why our metrics are indeed metrics for adaptability, in Section 5 we give extended uses for these metrics and Section 6 concludes this paper.

2 Development of Adaptability Metrics

Software can have several, sometimes an infinite number of different architectures. For example, the paper by Soni et al. [1] talks about four different software architectures: Conceptual architecture, Module architecture, Execution architecture and Code architecture. It is the opinion of the authors that complex software can have several different types, or levels, of architectures – thus there could be algorithmic architecture (that indicates the algorithms used in different modules); data structures architecture; document architecture (that indicates the documents or artifacts for the different modules); testing architecture; and so on.

Different architectures have varying degrees of adaptability (for each dimension of adaptability). For example, let Figure 1 depict the conceptual architecture for a system. In this system, data received from outside the system is received by the Communication ASIC Driver Block, which sends the received data (in the form of strings) to the Syntax Analysis Block. The Syntax Analysis Block parses the strings and sends the parsed code to the Semantic Analysis Block for further action. If the Semantic Analysis Block wishes to send data out then it sends the data directly to the Communication ASIC Driver Block. In Figure 1, any one of the

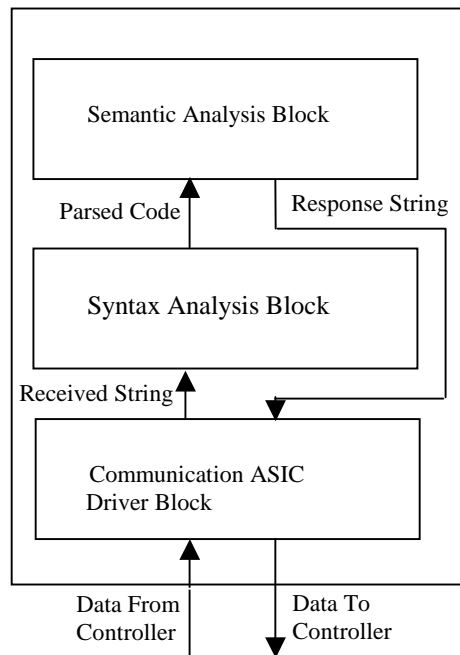


Figure 1: Conceptual Architecture for an Embedded System.

components or connections can be adaptable. Let it be assumed that the Syntax Analysis Block is adaptable. Then if one manifestation of this conceptual architecture is the architecture A1, given in Figure 2 (in this figure, the Embedded System's Higher Layers component does the work that the Semantic Analysis Block requires in Figure 1, the Output Manager handles the formatting of the outputs that the Embedded System's Higher Layers component wishes to send out of the system, the Device Driver component handles communication with the outside world, the Parser component handles the work of the Syntax Analysis Block of Figure 1, and the connections represented by block arrows may be message passing between the components) then it is possible that none of the components and connections of A1 is adaptable (this architecture could be construed as being statically adaptable – that is for any change of syntax, the parser will have to be changed).

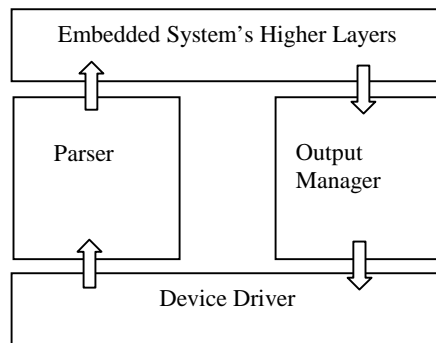


Figure 2: Architecture A1 : One Possible Implementation of Figure 1.

This paper attempts to develop a metric for adaptability of software. This is advantageous for the following reasons:

1. allows comparison of adaptability of different software.
2. allows to evaluate the degree of adaptability of a software.
3. allows modification to software to increase its adaptability (at the architecture level).

Before the adaptability metrics can be devised, certain precise definitions have to be made about adaptability and related concepts. Loosely speaking, software is adaptable if it can accommodate changes in its environment. However, environment can change in many ways or along different dimensions – for example, environment change may be the increase in the number of inputs to a software system from 20 inputs per second to 100 inputs per second (here the dimension is the number of inputs). Following definitions will make these concepts concrete (as mentioned in the Introduction, the following definitions are just one possible way to look at the NFR adaptability – there could be several other ways of viewing this NFR and we hope organizations choose the one that is most appropriate for the task at hand).

Dimension: any *attribute* external to the software system that impacts on the system – such as the inputs per second as in the above example, changes in business policy, etc.

Environment Change: change in the environment along *one* dimension.

Software Adaptability: software *accepts* environment change.

Thus Figures 1 and 2 were adaptable with respect to dimension “change in syntax” (the other dimensions could be character-set change, processing speed and the like).

3 Metrics for Adaptability

Any software architecture has two elements – components and connectors. Software could be adaptable with respect to either of these two elements on any of the architectures for that software. With respect to software adaptability, as defined above, adaptability of either element carries equal significance or weight. Adaptability of a component is in no way different from that of the adaptability of a connector. With this in mind the following definitions can be appreciated:

An adaptable element of architecture has a unit element adaptability index (EAI).
A non-adaptable element of architecture has zero EAI.

Architecture adaptability index (AAI) = $\frac{\text{EAI for all elements of architecture}}{\text{Total number of elements}}$.

Software adaptability index (SAI) = $\frac{\text{AAI for all architecture of the software}}{\text{Total number of architectures for that software}}$.

3.1 Examples

In Figure 1, only the Syntax Analysis Block component is assumed to be adaptable (with respect to syntax adaptation). All the remaining components and connectors are not adaptable (with respect to syntax adaptation). Thus, for Figure 1, the EAI's will be

$$\begin{aligned} \text{EAI (Syntax Analysis Block)} &= 1 \\ \text{EAI(for other elements)} &= 0 \\ \text{Total number of elements} &= 3 \text{ components} + 5 \text{ connectors} = 8. \end{aligned}$$

This is shown in Figure 3.

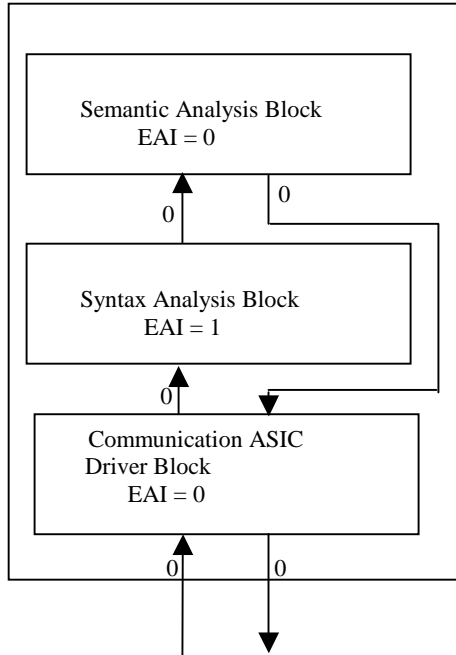


Figure 3: EAI's for Elements of Figure 1.

Hence,

$$AAI = 1/8 = 0.125.$$

Also, since we now have knowledge of only the conceptual architecture of Figure 1,

$$SAI = AAI = 0.125. \tag{1}$$

However, if architecture A1 is chosen, then since EAI for all elements is zero,

AAI (A1) = 0, but now

$$SAI (A1) = SAI1 = (0.125 + 0)/2 = 0.0625 \text{ or } 6.25\%. \tag{2}$$

In (2), we have knowledge of two architectures, that of Figure 1 and A1.

As another example, let us consider the architecture A2 given in Figure 4, which is yet another manifestation of the generic architecture of Figure 1 (this figure is identical to Figure 2, except that the parser is modifiable, hence adaptable with respect to syntax change). Here,

EAI (Dynamically Modifiable Parser) = 1

EAI (for all other elements and connectors) = 0

AAI (A2) = $1/8 = 0.125$ and (3)

SAI (A2) = SAI2 = $(0.125 + 0.125)/2 = 0.125 = 12.5\%$ (with respect to syntax adaptation). (4)

In (4), we again have the knowledge of two architectures, that of Figure 1 and A2.

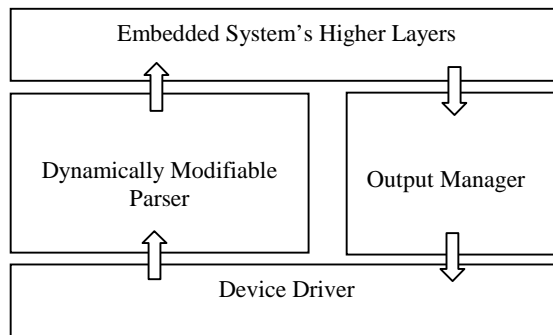


Figure 4: Architecture A2 : Another Implementation of Figure 1.

Comparing (2) and (4), the adaptability metrics let us conclude that software based on A2 is more adaptable than that based on A1, a fact that is evident intuitively as well.

3.2 Adaptability Metrification Process

The process of using our metrics is given in Figure 5.

As discussed earlier, the first step in using the metrics described in this paper is to define adaptability as suitable to the organization, project or the product. Once it is known by what yardstick a component of an architecture can be declared to be adaptable, then for each architecture, the AAI is calculated by determining the EAI for each element (components and connectors) of the architecture. The SAI is then calculated by taking the ratio of the sum of the AAI's to the number of architectures whose AAI's have been calculated. This is repeated for all architectures or as and when newer data regarding the software system becomes available.

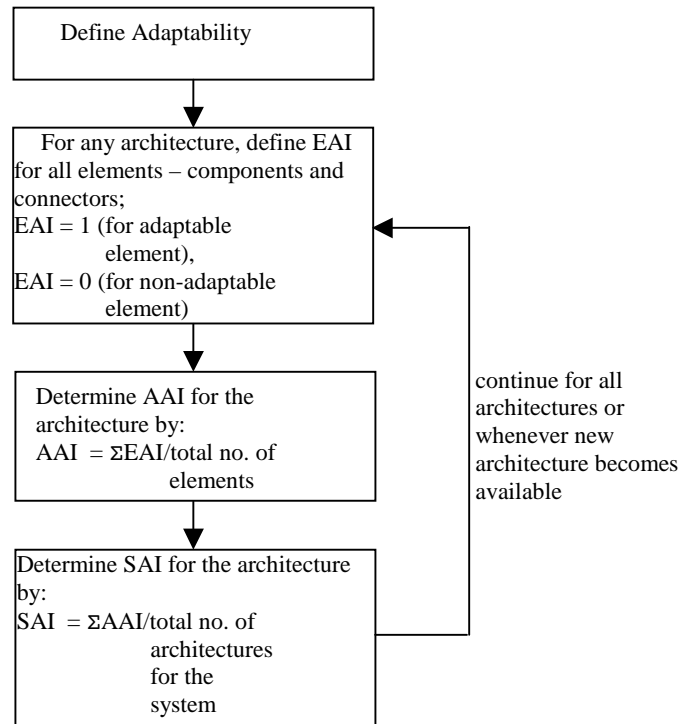


Figure 5: Process for Using the Adaptability Metrics.

3.3 Adaptability Over Dimensions

Usually when adaptability is considered it may not fit the narrow definition assumed here. Software may be adaptable over several dimensions; an element may be adaptable over several dimensions. In such cases the EAI's definition needs to be expanded as below:

$$\text{Total EAI} = \frac{\sum \text{EAI for each dimension of adaptability for the element}}{\text{Total number of dimensions.}}$$

The definitions for AAI and SAI remain the same as before.

Thus for example, if the system whose conceptual architecture is given in Figure 1, is adaptable not only towards syntax adaptation but also towards the protocol used for communication then there are two dimensions of adaptability. Then

$$\text{EAI}(\text{Syntax Analysis Block}) = \frac{1}{2} = 0.5 \text{ (since this component supports syntax adaptation)}$$

$EAI(\text{Communication ASIC Driver Block}) = \frac{1}{2} = 0.5$ (since this component supports protocol adaptation)

$EAI(\text{for others}) = 0$.

However,

$$AAI = (0.5 + 0.5)/8 = 0.125.$$

SAI depends on the module architecture, in this case, hence

$$SAI = AAI = 0.125.$$

However, if A2 is a manifestation of this extended Figure 1, then

$$AAI(A2) = 0.5/8 = 0.0625, \text{ and}$$
$$SAI = (0.125 + 0.0625)/2 = 0.09375 = 9.38\%,$$

which means that A2 is less adaptable than the extended Figure 1 (which is intuitively acceptable also).

3.4 Adaptability for Hierarchical Architectures

It is quite possible that a component or connector of an architecture could be further decomposed into sub-architectures, that is, there is an architecture hierarchy. If such a hierarchy exists, the scheme as discussed earlier remains valid. These sub-architectures for a higher-level component will be considered as separate architectures and their individual indices will be taken into account to get the SAI for the software.

For example, if in A2, the Dynamically Modifiable Parser component can be further decomposed as in Figure 6, then since only the Parser block of this figure is dynamically modifiable,

$$EAI(\text{Parser}) = 1$$
$$EAI(\text{others}) = 0$$

$$AAI(\text{Figure 6}) = 1/5 = 0.2, \text{ and} \quad (5)$$
$$SAI = (0.125 + 0.125 + 0.2)/3 = 0.15 = 15\% \text{ (from (1), (3) and (5)).}$$

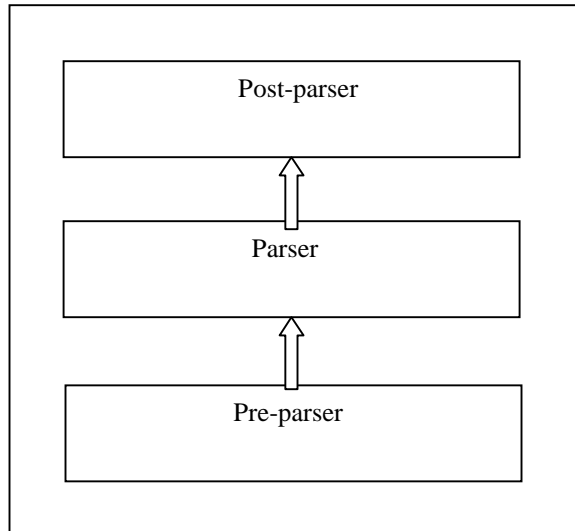


Figure 6: Decomposition of Dynamically Modifiable Parser Block Component of Figure 4.

This scheme lets further knowledge of software be used incrementally. Thus SAI changes as further knowledge of the software becomes available and lets the current adaptability index be known. This scheme also allows easy computation of the index and encourages decomposition to reveal further details of the software. Also this scheme does not tie the EAI of a component in any architecture to the EAI of its decomposition. This lets EAI's be computed independently and incrementally whenever further details become known.

3.5 Comparison of Adaptability of Architectures

The comparison using our metrics can take place at many levels:

1. element level – elements can be compared using the EAI's for each element.
2. architecture level – architectures can be compared using the AAI's for each architecture.
3. software level – software systems can be compared using SAI's for each system.

At all levels, the closer the index is to unity the more adaptable the corresponding item is. Thus at the element level, if an element has EAI of unity, it is adaptable, else it is not adaptable. At the architecture level, closer the AAI is to unity, the more adaptable the architecture is. Likewise at the software level, the closer the SAI is to unity, the more adaptable the software is. However, when comparing two SAI's it should be kept in mind that the number of architectures and their types considered in calculating the SAI's may be different. However, the closer either of

the SAI's is to unity, the more adaptable that software is; and more we know about the different architectures of software, the more precise its SAI will be.

3.6 Evolvability and Scalability of the Indices

The definition of the SAI in this paper lets the index grow as more knowledge of the system becomes available. Thus if at the beginning of the project, based on just the module architecture, an SAI of x was calculated, then towards the end of the project, based on knowledge of the code architecture, the SAI may be revised upward or downward to y (based on the adaptability of the code). Thus the indices can always be monitored to ensure that the software is being built with the expected adaptability.

SAI depends on the various architectures available for the software. This lets organizations choose which architectures they will include to calculate their indices. These decisions can be further modified for a project or even a product.

4 Properties of Adaptability Metrics

Section 2.2 of Fenton [3] discusses the basic requirements that any metric system should satisfy. First there should be an empirical relation system for the attribute adaptability. Secondly there should be a numerical relation system for the attribute. Thirdly the representation condition should be satisfied. Finally a scale of measurement should exist. On initial look it appears that all the four requirements are satisfied by our metric system as shown here.

The empirical relation system for the attribute adaptability can be given by the following:

$$E_1 = (C, E_R),$$

where, C is the set of software systems and E_R is the set of relations given by

$$E_R = \{R_1, R_2\},$$

where

Relation R_1 : $x \in R_1$ if x is adaptable

Relation R_2 : $(x, y) \in R_2$ if x is more adaptable than y .

The corresponding numerical relation system is given by

$$E_2 = (R, P)$$

where R is the set of real numbers and P is the set of relations over R given by

$$P = \{P_1, P_2\},$$

where

Relation P_1 : $x \in P_1$ if $x \in [0,1]$

Relation P_2 : $(x,y) \in P_2$ if $x > y$.

The mapping M from E_1 to E_2 is given by our formulas in Section 3 and in addition this mapping is a representation (as defined in Fenton [3]) because for each relation in E_1 there is a relation in E_2 - this is true for EAI and AAI but is *not* true for SAI in all cases. This is because the architectures used in computing SAI need not be the same for the two software systems being compared – however, if we impose the condition that the number and type of software architectures considered in calculating the SAI are to be the same, then SAI will also be a representation.

For determining the scale of measurement, we should note that all the indices were determined by the values given for the adaptable and non-adaptable elements of software architectures. If these values are changed, then we will get a different mapping from E_1 to E_2 , say M' , but M' will be related to M in a monotonic relation. Thus we conclude that the indices of adaptability that we have identified form an ordinal scale of measurement.

From the above it follows that the metrics that we have developed can indeed be considered metrics for adaptability (with the proviso on SAI as above).

We feel that having a set of adaptability metrics which obeys Cantor's theorem as suggested by Fenton [3] would be nice but it needs to be further investigated if adaptability metrics involving human level of confidence can be simplified to that extent in real life situations.

5 Extended Uses of the Adaptability Metrics

5.1 Legacy Systems

In many cases, especially legacy code systems, only the source code is available. In such cases using tools the code architecture may be obtained. For software systems that have *only* the code architecture,

$$SAI = AAI(\text{code architecture}).$$

5.2 Static vs. Dynamic Adaptation

For static adaptation, for each environment change a new software is developed. However, no software architecture has an adaptable component (for example, architecture A1 of Figure 2). Only the code architecture will be adaptable. Hence,

$$\text{SAI (for static adaptation)} = \frac{\text{AAI (code architecture)}}{\text{Total No. of Architectures}}$$

However, for dynamic adaptation, at least one component will be adaptable (for example architecture A2 of Figure 4) and as such the usual formula should be used:

$$\text{SAI(for dynamic adaptation)} = \frac{\sum \text{AAI}}{\text{Total No. of Architectures.}}$$

6 Conclusion

We have in this paper developed three adaptability indices for software architectures –

1. element adaptability index
2. architecture adaptability index
3. software adaptability index.

All these indices have values between 0 and 1 – a value of 0 indicates that the architecture is not adaptable at all, while a value of 1 indicates that the architecture is fully adaptable for the chosen dimensions. We have also shown how these metrics for software adaptability are evolvable and scalable.

We hope that these metrics for software adaptability will be useful to industry and will be used as one of the external and internal metrics as defined in the ISO/IEC 14598 [8] set of standards.

As mentioned earlier, this paper represents our initial investigation on measuring adaptability. However, there is a lot of work still to be done – this is because we feel that many NFRs fulfill the notion of “satisficing” and not of “satisfying” [9]. The notion of satisficing means that NFRs can only be satisfied within acceptable limits and not absolutely, by any software. We are working on extending the definition of adaptability metrics to include this notion of satisficing and we expect this extension to give us even further insights into the problems associated with measuring adaptability. We are also exploring the concept of negative adaptability which will let us perform tradeoff analysis with respect to the NFR Framework and lead us to a more intuitive set of metrics (we have partially accomplished some of these goals in [10]).

References

1. Soni D, Nord RL, Hofmeister C, Software Architecture in Industrial Applications, Proceedings of the 17th International Conference on Software Engineering, 1995, Seattle, Washington, **pp** 196-207.
2. Subramanian N and Chung L, Architecture-Driven Embedded Systems Adaptation for Supporting Vocabulary Evolution, Proceedings of International Symposium on Principles of Software Evolution, November, 2000, Kanazawa, Japan, **pp** 144-153.

3. Fenton NE, Software Metrics – A Rigorous Approach, Chapman & Hall, London, 1991.
4. Gilb T, Principles of Software Engineering Management, Addison Wesley, England, 1988.
5. Duenas JC, de Oliveira WL, de la Puente JA, A Software Architecture Evaluation Model, In: Frank van der Linden (ed) Development and Evolution of Software Architectures for Product Families. Springer-Verlag, Heidelberg, 1998, **pp** 148-157 (Lecture Notes in Computer Science no. 1429)
6. Shaw M and Garlan D, Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, New Jersey, 1996.
7. Bass L, Clements P and Kazman R, Software Architecture in Practice, SEI Series in Software Engineering, Addison-Wesley, Massachusetts, 1998.
8. ISO/IEC 14598-1 International Standard, Information Technology – Software Product Evaluation, 1999.
9. Chung L, Nixon BA, Yu E, and Mylopoulos J, Non-Functional Requirements in Software Engineering, Kluwer Academic Publishers, Boston, 2000.
10. Chung L and Subramanian N, Process-Oriented Metrics for Software Architecture Adaptability, submitted for publication.