Automated Analysis of Software Designs with Graphic Quality Models

QIAN ZHANG
Department of Computer Science
National University of Defense Technology
Changsha, CHINA
Email: zhangqian@nudt.edu.cn

HONG ZHU
Department of Computing
Oxford Brookes University
Oxford OX33 1HX, UK
Email: hzhu@brookes.ac.uk

Abstract: Software quality models play a significant role in software quality assurance. Based on our previous work on graphic modelling of software quality, this paper extends the quality modelling language to enhance its expressiveness and to facilitate automated analysis of software quality as designed. A collection of algorithms that are implemented in an automated tool for the analysis of software quality are presented and illustrated by examples.

Keywords: Software quality, Automated software tools, Analysis of software quality, Modelling

1 Introduction

Software quality models play a significant role in the quality assurance of software development [1]. Existing software quality models can be classified into two types. Hierarchical models, such as McCall model [2], Boehm model [3], ISO model [4], and the more recent Bansiya and Davis' model of OO software design [5], define a set of quality related properties and organises them into a hierarchical structure to express the positive relationships between them. However, they are incapable of expressing negative relations between quality related properties. A relational model usually defines a number of stereo types of relationships between quality attributes, such as positive, negative and neutral relations. Typical examples of such quality models include Perry Model [6], Gillies Model [7, 8]. These quality models can help software developers to improve software quality by providing guidelines to software development activities, such as in the elicitation of quality requirements. However, as pointed out by Dromey [9, 10], they fail to take software structures into account. Due to the stereo typing of relationships between quality attributes, they are also incapable to deal with complicated relationships between quality attributes. They provide little help to the design of software systems.

Addressing these problems, in [11, 12, 13], we proposed the HASARD method to software quality modelling and analysis in order to provide more instructive information about software designs. A diagrammatic notation for software quality modelling was devised [11]. A method for deriving software quality models from architectural designs was developed [13]. A software tool called SQUARE for software quality modelling and analysis was designed and implemented [13]. In this paper, we further develop the technique by focusing on automated analysis of software quality with such graphic models.

The paper is organized as follows. Section 2 briefly reviews the graphic notation for representing software quality models and extends it with facilities that enable automated analysis of quality models. Section 3 discusses the methods and algorithms for automated analysis of quality related issues in software designs. Section 4 concludes the papers with remarks on the current state of the implementation of the automated analysis tools for quality analysis and case studies of the method.

2 Graphic Quality Models

As shown in Fig. 1, a quality model in our graphical modelling notation is a directed graph, which consists of a set of *nodes* and a set of *links* between the nodes.

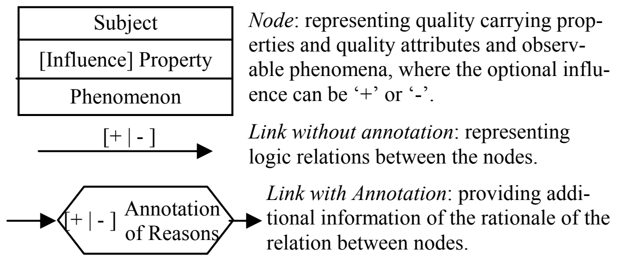
Subject
[Influence] Property
Phenomenon
Node: representing quality carrying properties and quality attributes and observable phenomena, where the optional influence can be '+' or '-'.
[+ | -]
Link without annotation: representing logic relations between the nodes.
[+ | -]
Annotation of Reasons
Link with Annotation: providing additional information of the rationale of the relation between nodes.

Fig. 1. Graphical notation of quality models

Each node represents a quality related property of the software system. Such a property may demonstrate itself by a certain phenomenon of a certain component or subsystem of the system or the whole system. To specify such a property, a node in the graphic quality modelling language contains three parts in the form of compartments. The *phenomenon compartment* describes a particular observable phenomenon in the system. The *subject compartment* specifies the entity in the system that demonstrates the phenomenon. The element in the subject compartment can be a component or a connector of the software system, or a subsystem even the system itself, or an external entity of the system, etc. The *property compartment* gives a classification of the phenomenon in terms of a *quality carrying property*, which can be a quality attributes such as correctness or a property that affects the quality of the system somehow such as the size of the component. In addition to the classification of the phenomenon by a quality attribute or quality related property, the relationship between the phenomenon and the property could be positive or negative. A phenomenon is *positive* to a quality property, if the phenomenon is observed, then the entity is good at the property. Otherwise, the phenomenon is negative to the quality property, i.e. the observation of the phenomenon implies the entity is poor at the property. Such relationships between phenomena and properties are specified in the *influence factor*. It is by default positive. Positive influence is denoted by the symbol + and negative influence is denoted by the symbol '-'. This influence factor is a new facility introduced in this paper.

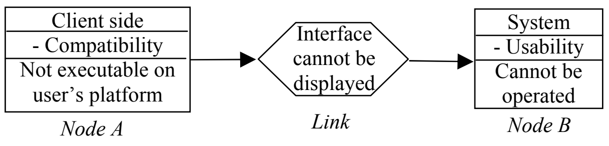

Fig. 2. Example of nodes and link

For example, in Fig. 2, the node *A* specifies a phenomenon that the client-side subsystem of a web-based application in the client-server architecture is not executable on the user's platform. It states that this phenomenon is classified to the compatibility issue. If the phenomenon is observed, then the compatibility of the system is poor. Thus, the influence is negative.

The *links*, i.e. directed arcs, between the nodes specifies the relationships between the phenomena specified by the nodes. Such a relationship can be either *implication* or *prohibitive*. It is also called *influence factor* of the link. An implication relationship from node *A* to node *B* means that the observation of the phenomenon on node *A* implies the occurrence of the phenomenon on node *B*. Such an implication can be logic causal relation, enabling condition, or plausible result. A prohibitive relationship from node *A* to node *B* means that the observation of the phenomenon on node *A* will prevent the occurrence of the phenomenon on node *B*. Such a prevention relationship can also be either logically disabling the phenomenon of node *B* to happen or the phenomenon of node *B* will plausibly not likely to happen. Each link may contain an optional annotation for the reasons why the two nodes are related. The implication relationships are denoted by the symbol '+' while prohibitive relationships are denoted by the symbol '-'. By default, a link has the implication relationship if it is not explicitly specified. The influence factor of link is also a new facility introduced in this paper. Together with the influence factor of the nodes, it not only significantly improved the expressiveness of the quality modelling language, but also provided a crucial facility for automated analysis of quality models.

For example, in Fig. 2, the link between nodes *A* and *B* states that if the client-side code cannot be executed on the user's platform, the system cannot be operated, because the interface cannot be displayed on the user's screen at all. In most cases, the reasons are self-evident and obvious. Therefore, it provides a facility for human validation of the quality model.

Fig. 3 shows an example of quality model in the graphic notation. It will be used through this paper to illustrate the analysis of software quality.

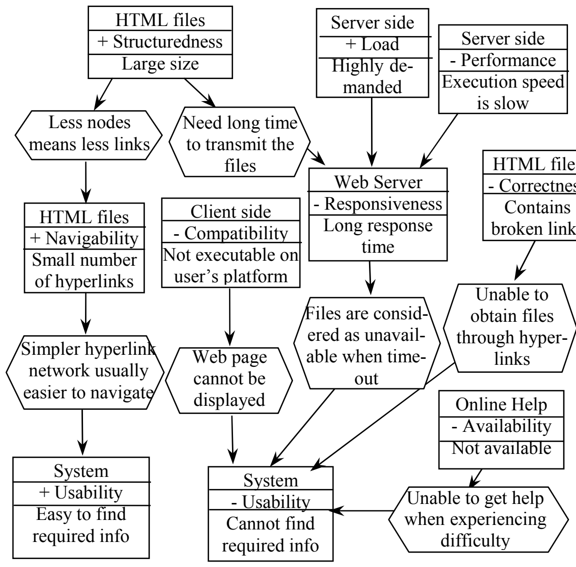
HTML files
+ Structuredness
Large size
Server side
+ Load
Highly demanded
Server side
- Performance
Execution speed is slow
Less nodes means less links
Need long time to transmit the files
Web Server
- Responsiveness
Long response time
HTML files
- Correctness
Contains broken links
HTML files
+ Navigability
Small number of hyperlinks
Client side
- Compatibility
Not executable on user's platform
Files are considered as unavailable when timeout
Unable to obtain files through hyperlinks
Simpler hyperlink network usually easier to navigate
Web page cannot be displayed
Online Help
- Availability
Not available
System
+ Usability
Easy to find required info
System
- Usability
Cannot find required info
Unable to get help when experiencing difficulty

Fig. 3. An example of quality model

3 Automated analysis algorithms

In this section, we discuss various tasks of quality analysis at software design stage and how they can be supported by automated tools. The algorithms for such tool support are presented.

3.1 Contribution factors

In the analysis of software architectural designs, we often want to know how a quality issue is addressed. We want to know which components, connectors or the properties of the configuration are related to the quality issue and how they collectively provide the solution to meet quality requirements. The contribution factors of a quality attribute is a set of properties of the components and/or connectors and the configuration of the architecture that affect the quality issue according to the design. For example, consider the quality model given in Fig. 3. We can derive the sub-graph shown in Fig. 4 for the contribution factors of a server's responsiveness.

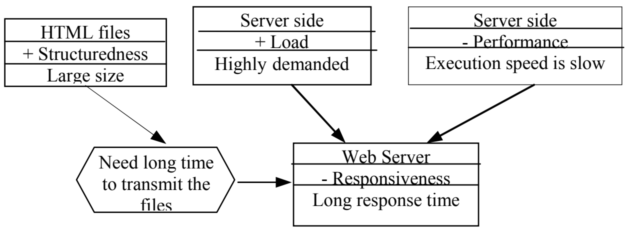

Fig. 4. Factors of server's responsiveness

This quality analysis task can be supported by our tools reported in [13], which implements the following algorithm.

```
ALGORITHM   A1 (* Derivation of contribution factors
to a quality attribute. *)
INPUT:
  QualityModel = < NodeList, LinkList >;
    (* the quality model is represented as a graph that
    consists of a set NodeList of nodes and a set LinkList
    of links. *)
  Component; (* the name of the component *)
  QualityAttribute; (* the quality attribute *)
OUTPUT:
  RelatedNodeList; (* the set of nodes in the quality
model related to the quality attribute*)
  RelatedLinkList; (* the set of links in the quality model
related to the quality attribute *)
BEGIN
  RelatedNodeList := { };
  RelatedLinkList := { };
  FOR each node N in NodeList DO
     IF (N's component name = Component)
        AND (N's property = QualityAttribute)
     THEN add N into RelatedNodeList;
  END_FOR;
  REPEAT
     FOR each link L in LinkList DO
       BEGIN
         IF (L's head is in RelatedNodeList)
            AND (L's head is not equal to L's tail)
        THEN
            IF L is not in RelatedLinkList
            THEN Add link L to RelatedLinkList;
            IF L's tail is not in RelatedNodeList
            THEN Add L's tail to RelatedNodeList;
        END_IF
       END;
  UNTIL no more element is added into RelatedLinkList
          or RelatedNodeList;
  OUTPUT RelatedLinkList and RelatedNodeList;
END_ALGORITHM.
```

3.2 Impacts of design decisions

Another frequently asked question in the analysis of a software architectural design is "what are the consequences of a design decision on the properties and functionality of a component or connector?' In such cases, we need to find out what are the quality attributes that are affected by the design decision. Such information can also be derived from a well constructed quality model. For example, consider the quality model depicted in Fig. 3. We can obtain the sub-graph shown in Fig. 5 that represents the impacts of the quality carrying property of HTML files' size on other quality attributes. It shows that the size of HTML files affects the navigability and responsiveness of the system, which further affects the usability of the whole system.

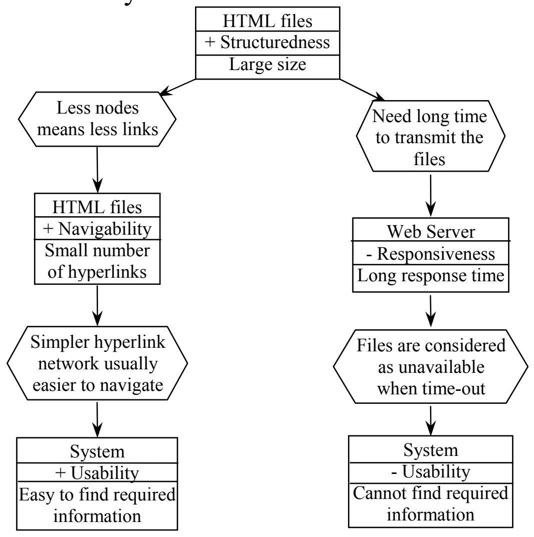
HTML files
+ Structuredness
Large size
Less nodes means less links
Need long time to transmit the files
HTML files
+ Navigability
Small number of hyperlinks
Web Server
- Responsiveness
Long response time
Simpler hyperlink network usually easier to navigate
Files are considered as unavailable when time-out
System
+ Usability
Easy to find required information
System
- Usability
Cannot find required information

Fig. 5. Affected attributes by size of HTML files

This analysis task can also be automated. The following is the algorithm that has been implemented in our automated tools.

```
ALGORITHM A2 (* Derivation of the impact of a com-
ponent's property. *)
INPUT:
  QualityModel = < NodeList, LinkList >;
   Component (* the name of the component *),
   QualityAttribute (* the property of the component *),
OUTPUT:
   EffectedNodeList (* the set of nodes in the quality
model effected by the component's property *);
   EffectedLinkList (* the set of links in the quality model
effected by the component's property *);
BEGIN
  EffectedNodeList := { };
  FOR each node N in NodeList DO
    IF (N's component name = Component)
       AND (N's property = QualityAttribute)
    THEN add N into EffectedNodeList;
  END_FOR;
  EffectedLinkList := { };
  REPEAT
```

```
        FOR each link L in LinkList DO
          BEGIN
            IF (L's tail is in EffectedNodeList)
                AND (L's head is not equal to L's tail)
            THEN
              IF L is not in EffectedLinkList
              THEN Add link L to EffectedLinkList;
              IF L's head is not in EffectedNodeList
              THEN Add L's head to EffectedNodeList;
            END_IF
          END
        END_FOR;
    UNTIL no more element is added into EffectedLinkList
            or EffectedNodeList;
    OUTPUT EffectedLinkList and EffectedNodeList;
END_ALGORITHM.
```

3.3 Quality risks

A design decision may have positive as well as negative effects on a quality attribute. The negative effects may impose quality risks to the system. Therefore, it is often desirable to know where the quality risks are within an architectural design. This can also be derived from a quality model.

A negative effect of a design decision can be recognised by searching for the links and nodes in the quality model that have a negative effect on the quality attribute. Such a negative effect could be in one the following two forms. First, there is a negative influence factor in the node while there is a positive influence factor on the link. Second, there is a negative influence factor on the link while there is a positive factor on the node. In the former, a phenomenon will be observed that means the quality attribute will be worse. In the later case, the phenomenon that indicates a better value of the quality attribute will be prohibited to happen. For example, in the quality model depicted in Fig. 3, there is a link between the node *HTML files* with the property of *large size* and the node *web server* with a property of *responsiveness*. There is a positive influence factor marked on the link between the large size of HTML file and the phenomenon of 'long response time'. This is because that the larger the HTML file size is, the longer the response time will be. Because the phenomenon of long response time has a negative influence factor on usability, the large file size has a negative effect on usability. Therefore, a design decision of large file size is a risk to the quality attribute of responsiveness. The further consequences of a quality risk can be identified and analyzed. In certain cases, a negative effect, i.e. a quality risk, is not the consequence of a single design decision. Instead, it can be the consequence of a number of other design decisions. In that case, all the causes must be identified so that a better design can be made. This can also be derived from graphic quality models by using the following algorithms.

```
ALGORITHM   A3 (* Derivation of design decisions
which have risks to the system's quality. *)
INPUT:
   QualityModel = < NodeList, LinkList >;
OUTPUT:
  RelatedNodeList (* the set of nodes in the quality model
related to risk rising decisions *);
BEGIN
  RelatedNodeList := { };
  FOR each node N in NodeList DO
     IF (N's influence factor is negative)
     THEN add N into RelatedNodeList;
  END_FOR;
  OUTPUT RelatedNodeList;
END_ALGORITHM.
```

3.4 Relationships between quality issues

An important question to be answered in quality analysis is the interrelationship between two quality issues. For example, how server's performance is related to the system's usability? Answers to such questions can be found from the quality model by searching for all paths from a node that represents one quality issue to the nodes that represents the other quality issue. The algorithm used to implement the supporting tool is given below.

```
ALGORITHM   A4 (* Derivation of Relationships be-
tween quality issues *)
INPUT:
  QualityModel = < NodeList, LinkList >;
  Component1 (* the name of the first component *),
  Component2 (* the name of the second component *),
  QualityAttribute1 (* the first quality attribute *),
  QualityAttribute1 (* the second quality attribute *),
OUTPUT:
  RelatedNodeList (* the set of nodes in the quality model
on the paths between two quality attributes*);
  RelatedLinkList (* the set of links in the quality model
on the paths between two quality attributes *);
BEGIN
  RelatedNodeList := { };
  RelatedLinkList := { };
  Node1:=NULL;
  Node2:=NULL;
  TemptNodeList:= { };
  TemptNode:=NULL;
  FOR each node N in NodeList DO
     IF (N's component name = = Component1)
        AND (N's property = = QualityAttribute1)
     THEN Node1=N;
     ELSE IF (N's component name = = Component2)
               AND (N's property = =QualityAttribute2)
     THEN Node2:=N
     END_IF;
   END_FOR;
   Add Node1 to TemptNodeList;
```

```
    CurrentNode :=Node1;
    Search(CurrentNode, Node1, Node2, QualityModel,
           TemptNodeList, RelatedLinkList,
           RelatedNodeList);
   OUTPUT RelatedLinkList and RelatedNodeList;
END_ALGORITHM.
```

In the algorithm A4, the following function of depth first search is used.

```
FUNCTION Search (Component, Component1,
Component2, QualityModel, CurrentNodeList,
ResultLinkList, ResultNodeList)
 (* Depth-First Search *)
 BEGIN
      TemptNode=NULL;
      FOR each link L in LinkList that
                 L's head = = Component DO
         BEGIN
             Add L's tail to CurrentNodeList;
             IF L's tail= =Component2
             THEN  (* Find a path and record it*)
                  TemptNode=L's tail;
                  REPEAT
                      Add TemptNode to ResultNodeList;
                      TemptNode =
                             TemptNode's previous node
                              of CurrentNodeList;
                       Add link TemptL( whose head = =
                                    TemptNode's Next node of
                                    CurrentNodeList AND
                                     whose tail==TemptNode)
                      to ResultLinklist;
                   UNTILL TemptNode= =Component1;
                   Remove L's tail From CurrentNodeList;
              ELSE  (* Depth first *)
                  Search (L's tail, Component1,
                          Component2, QualityModel,
                          CurrentNodeList, ResultLinkList,
                          ResultNodeList);
               END_IF
      END_FOR;
      remove Component from TemptList;
END_FUNCTION
```

3.5 Trade-off points

In many situations, a quality risk cannot be resolved without compromising on other quality issues because these quality issues are conflicting with each other. In such cases, trade-offs between the quality attributes must be made and a balance between them must be achieved through appropriate design decisions.

For example, consider the quality model depicted in Fig. 3. The size of HTML files positively affects the navigability of the hypertext network, but negatively affects responsiveness of the web server. Therefore, navigability is in conflict with responsiveness. A trade-off between them must be made so that responsiveness is within a tolerable range while navigability is also acceptable. Such a trade-off occurs in the form of deciding on a suitable size of HTML file. In other words, HTML file size is a trade-off point.

From this example, we can see that a trade-off point is a node in the quality model that has a negative effect to one or more quality attributes and at the same time it has positive effects on one or more quality attributes. Trade-off points can also be derived from quality models automatically. The algorithm that used to implement this is given below.

```
ALGORITHM    A5 (* Derivation of trade-off points *)
INPUT：
   QualityModel = < NodeList, LinkList >;
OUTPUT：
   RelatedNodeList (* the set of trade-off points *);
BEGIN
   RelatedNodeList={};
   TemptNodeList={};
   TemptNodeList:= result from calling A3;
   FOR each node N in TemptNodeList DO
      FOR each link L in LinkList
             AND ( L's head= =N OR L's tail = = N) DO
         IF ( L's head !=N
             AND (L's head's influence factor
                   = = L's influence factor))
         THEN Add L's head to RelatedNodeList;
         IF (L's tail !=N
             AND (L's tail's influence factor
                   = = L's influence factor))
         THEN Add L's tail to RelatedNodeList;
      END_FOR
    OUTPUT RelatedNodeList;
END_ALGORITHM
```

To make a right decision on a trade-off point, we need to understand all the consequences of the design decisions. In certain complicated situations, a trade-off point is a consequence of a number of other design decisions. Once a trade-off point is recognised, we can derive all quality attributes that the trade-off point affects, and to find all the factors that affect the trade-off point using the algorithms and the automated tools as discussed above.

4 Concluding remarks

The main contributions of this paper are two folds. First, the graphic quality modelling language presented in [11,12] is extended to enhance its expressiveness and to facilitate automated analysis of software quality according to its design. Second, various quality analysis tasks at design stage are recognised. Algorithms to automate these quality analysis tasks are presented. An automated software tool called SQUARE has been developed [13]. It implemented the algorithms for the analysis of graphic quality models. Figure 6 shows its overall structure.

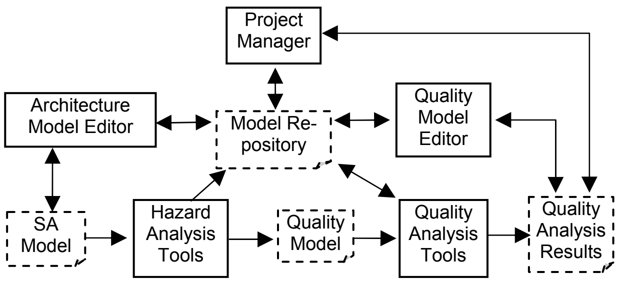

Figure 6. The structure of SQUARE tool

In comparison with existing methods of evaluation and analysis of software architectures, such as scenario-based methods such as SAAM, ATAM and others [14–21], our approach can be performed systematically with strong support by automated tools as shown in this paper. Case studies of the method and the tool have been conducted. However, for the sake of space, the case studies will be reported separately.

We are further investigating how quality models can be constructed and validated. In [11], a hazard analysis method was proposed, which is further developed in [12, 13]. It is worth studying how the method can be combined with scenario-based methods, for example, by representing the results of scenario analysis in our graphic quality models, and/or by deriving quality model using scenarios.

Acknowledgement

This work reported in this paper is partly supported by 863 Hi-Tech Program of China under the Grant No.2002AA116070.

References:

[1] B. Kitchenham, and S. L. Pfleeger, Software Quality: The Elusive Target, *IEEE Software*, Vol. 13, No.1, 1996, pp.12-21.
[2] J. McCall, P. Richards, and G. Walters, Factors in Software Quality, *Technical Report* CDRL A003, US Rome Air Development Centre, Vol.1, 1977.
[3] B.W. Boehm, J. Brown, H. Kaspar, M. Lipow, G. MacLeod, and M. Merrit, Characteristics of Software Quality, *TRW Series of Software Technology*, Vol. 1, North-Holland, 1978.
[4] International Organisation for Standardization, *ISO 9126: Information Technology -- Software Product Evaluation -Quality Characteristics and Guidelines for Their Use*, ISO, 1992.
[5] J. Bansiya, and C. G. Davis, A Hierarchical Model for Object-Oriented Design Quality Assessment, *IEEE TSE*, Vol.28, No.1, 2002, pp.4-17.
[6] W. E. Perry, *Quality Assurance for Information Systems: Methods, Tools and Techniques*, John Wiley & Sons, 1991.
[7] A. Gillies, Modelling Software Quality in The Commercial Environment, *Software Quality Journal*, Vol.1, 1992, pp.175-191.
[8] A. Gillies, *Software Quality: Theory and Management*, 2nd Edition, International Thomson Computer Press, 1997.
[9] R. G. Dromey, A Model for Software Product Quality, *IEEE TSE*, Vol.21, No.2, 1995, pp.146-162.
[10] R.G. Dromey, Cornering the Chimera, *IEEE Software*, Vo.13, No.1, 1996, pp.33-43.
[11] H. Zhu, Y. Zhang, Q. Huo, and S. Greenwood, Application of Hazard Analysis to Software Quality Modelling, *Proc. of COMPSAC'02*, IEEE CS, 2002, pp.139-144.
[12] H. Zhu, *Software Design Methodology: From Principles to Architectural Styles*, Elsevier, 2005.
[13] Q. Zhang, J. Wu, and H. Zhu, Tool Support to Model-based Quality Analysis of Software Architectures, *Proceedings of COMPSAC'06*, IEEE CS, 2006. (In press)
[14] C.-H., Lung, *et al.*, An approach to software architecture analysis for evolution and reusability, *Proc. of CASCON*, 1997.
[15] P. Bengtsson, and J. Bosch, Scenario-based software architecture reengineering, *Proc. of ICSR5*, IEEE CS Press, 1998, pp.308-317.
[16] N., Lassing, D., Rijsenbrij, and H., van Vliet, Towards a broader view on software architecture analysis of flexibility, *Proc. of APSEC'99*, IEEE CS Press, 1999, pp.238-245.
[17] P., Bengtsson, *Architecture-Level Modifiability Analysis*, Ph.D. Thesis, Blekinge Institute of Technology, Sweden, 2002.
[18] E., Folmer, J.V. Gurp, and J. Bosch, Scenario-based assessment of software architecture usability, *Proc. of Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction*, ICSE 2003. Portland, Oregon.
[19] P., Bengtsson, N., Lassing, J., Bosch, and H. van Vliet, Architecture-Level Modifibility Analysis, *The Journal of Systems and Software,* Vol. 69, 2004, pp.129-147.
[20] B. Tekinerdogan, ASAAM: aspectual software architecture analysis method, *Proc. of the Fourth Working IEEE/IFIP Conference on Software Architecture* (*WICSA'04*), 2004.
[21] S. Aier and M. Schönherr, Evaluating integration architectures – A scenario-based evaluation of integration technologies, *Lecture Notes in Computer Science*, Vol. 3888, 2006, pp.2.