

Towards Quantitative Evaluation of UML based Software Architecture

Jinhua Li, Zhenbo Guo, Yun Zhao, Zhenhua Zhang, and Ruijuan Pang
College of Information Engineering, Qingdao University
qduli@126.com

Abstract

The architecture of a software system is a critical artifact in the software lifecycle and should be evaluated as early as possible. Recent efforts to software architecture evaluation are concentrated on scenario-based methods which are qualitative, subjective and need not any special architecture description languages. This paper investigates an approach to metrics based quantitative evaluation of UML software architecture. UML is a visual modeling language with well-formed hierarchical syntax and semantics, and is uniformly applied in various development stages. With supplementation UML has been adapted to describing software architecture. By utilization of these features three types of metrics for UML diagrams are proposed. They measure the amount of information, visual effect and connectivity degree in different UML diagrams. The application of these metrics in quantitative evaluating qualities at the architecture-level such as system scale, complexity and structural characteristics is discussed.

1. Introduction

As the roles of software architecture in design decisions and software qualities increase, it is important to evaluate architecture at early the stages of software development as possible [1, 2]. The evaluation of software architecture belongs to one of the most researched topics since the discipline of software architecture emerges [3]. This paper presents our initial work on the quantitative evaluation of software architecture based on UML notation [4]. After a brief introduction of software architecture and architecture description languages, related works is overviewed in section 2. Section 3 describes a set of metrics based on UML models, which utilizes the graphical expression, multiple viewpoints, and formal definitions of UML. The applicability of these metrics in architecture quality evaluation is analyzed

theoretically and showed with a case study in section 4. Section 5 concludes this article with the future works.

2. Preliminaries

2.1. Software architecture and ADLs

Software architecture that determines the system's organization, overall structure and global behavior should be described from various viewpoints [5, 6]. Software architecture models usually consist of components, connectors, configuration, ports and roles. Components are reusable software objects performing the functions of data process or data storage. Components communicate with environments via interfaces that are composed of a set of ports. Connectors describe the interactions between components. The interface of a connector consists of a set of roles which define the actors participating in the communication. The configuration defines logical restrictions and structural constraints on the components and connectors in the software architecture.

Different architecture description languages (ADLs) have been designed to document software architecture. ADLs can be grouped into two categories: the specially designed ADLs such as C2, Unicon or ACME[6], and those ADLs based on UML[7, 8, 9]. The specific ADLs are usually textual languages, mainly address certain aspects of software architecture, and have specific applications.

Figure 1 shows an architecture configuration with extended UML [6]. The system IS2000 is composed of three components Acquisition, Imaging, Exporting, and two connectors Control and Data. IS2000 as a component have ports for interactions with the outside world. The shadowed rectangle represents component. Its ports are shown as small black squares at the edge of the component, with the port name near the port. An elongated hexagon represents a connector, roles of which are small black circles at the edge of the connector. A binding between ports is shown in UML

association (a line), as a connection between a port and a role.

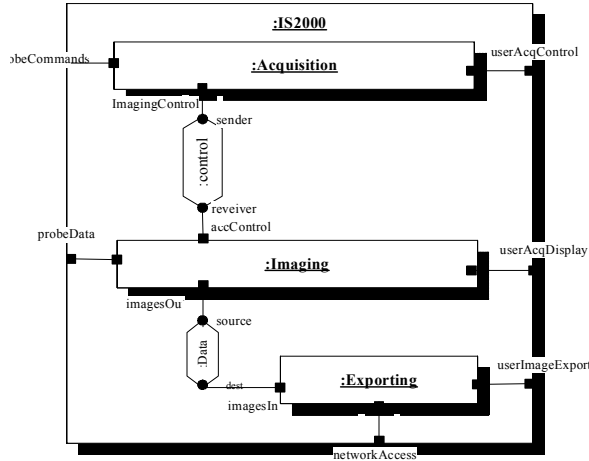


Figure 1 A software architecture configuration [6]

2.2. Software architecture evaluation

The proposed techniques for software architecture evaluation can be divided into two classes: questioning techniques and measuring techniques [1, 2]. Most of questioning techniques use scenarios, since they are very concrete; enable detailed analysis and statements about their impacts. Precise description of software architecture is not necessary. However, scenarios-based methods evaluate architecture qualitatively. The results are heavily dependent on the selected set of scenarios, and the subjective interpretation related to the given quality attributes. Besides, the methods can not determine the minimum number of scenarios.

Metrics-based techniques may evaluate quality attributes of architecture more precisely and objectively [10]. The work in [11] describes two metrics called information coupling and cohesion which are designed for measuring various aspects of information exchanges in the software architecture. Measurements for software quality attributes such as risk, adaptability and stability at architecture-level are much investigated [12, 13, 14]. These works do not consider the precise description of software architecture. Object-oriented metrics is also adapted to evaluating the architectural modifiability [15]. Similarly, measuring object-oriented features at the architectures level from UML diagrams is reported [16]. The metrics includes a number of basic UML elements such as classes, states, associations, transitions, and the depth of class inheritance. However, the metrics do not directly reveal the system characteristics at the architecture level.

Comparatively, our work has two different features. Firstly, the measurements are specially designed for UML models by considering the graphical and structural characteristics, and from multiple views of a system. Secondly, the metrics measures different aspects of a system as a whole from different perspectives, as well as the basic modeling elements, and thus can indicate various quality attributes in software architecture.

3. A set of UML metrics

Three basic metrics for various UML diagrams are defined: Information Content (IC), Visual Effect (VE) and Connectivity Degree (CD). IC tries to measure the amount of architecture information from UML diagram. VE expresses the optical impact of UML diagrams on human beings. CD indicates the dependency and complexity of software architecture.

3.1. Information content

The measurement IC defines how much design information a UML diagram expresses. UML provides a large number of graphical modeling elements for different software artifacts. Most of UML tools are also provided with textual notes as comments on the graphical elements. Each kind of these elements in the diagram contains different amount of design information. Take Figure 1 for example. The graphical elements modeling component, connectors and their relations describe the system structure and organization, and they contain the essential information of the software architecture. Whereas the ports and roles modeled as stereotyped classes represent little content at the architecture-level. If one wants to understand or to modify the software architecture, one has pay more attention on components, connectors and their composition or binding relations than on the ports of components or roles of connectors. This means that different graphical elements serve quite different in modeling software architecture.

In order to determine the value of information content, we reorganize the UML graphical elements in a hierarchy based on the UML meta-model, place elements that play an important role in modeling software architecture at high levels, those elements expressing less meaning at lower levels. Figure 2 shows the hierarchy of graphical elements in UML structure diagrams. The UML package serving as structural modeling element can encapsulate classes, interfaces, and relationships. A package can also be contained in another package. Considering the

important role of comments in the software development, class notes are placed below the level of class in the hierarchy.

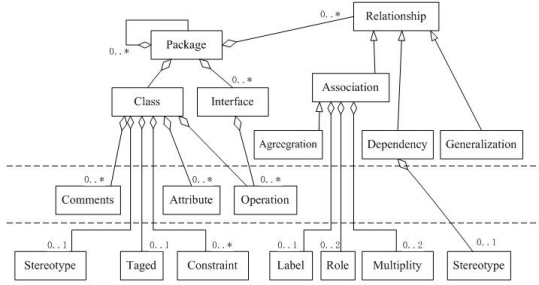


Figure 2 Hierarchy of modeling elements in structure diagrams

Definition 1.1: IC of a UML diagram (an architecture view) is defined as the sum over the number of modeling elements times their weights. Let E be UML modeling elements, function $weight(E)$ the weighted value of element E at the appropriate level in the hierarchy, then IC of a UML diagram with n elements is

$$IC(diagram) = \sum_{i=1}^n E_i \times weight(E_i) \quad (1.1)$$

The greater of IC is, the more complexity is the software architecture, and the more effort will be taken in the software development. Different kinds of modeling elements express different amount of design information in different details, and they are placed at different levels in the hierarchy. Modeling elements at the higher level have much effect on the software system, and thus are assigned to larger weighted values. When calculating the number of associations, multiple relations are transformed into binary relations. Relations are considered to have directions, too.

For instance, there are 6 classes, 10 associations and 20 stereotypes in Fig.1. If the weighted values at level 1, level 2 and level 3 are supposed to be 1, 0.75 and 0.5 respectively, then the value IC in Fig.1 is $6+10+0.5 \times 20=26$.

Analogy to Fig. 2, hierarchical graphs can be constructed for other UML diagrams, such as sequence diagram, state diagram and deployment diagram. The calculation of IC for such UML diagrams is similar.

Definition 1: IC of an architecture with different views described in UML diagrams is defined as follows.

$$IC(architecture) = \sum_{diagram \in architecture} \frac{IC(diagram)}{count_of_diagram(architecture)} \quad (1)$$

3.2. Visual effect

Because UML is a graphical modeling notation, the optical effect of graphs on human beings should be taken into consideration when the architecture of a software system is documented in UML diagrams. VE measures to what extent that a UML diagram affects human understands of software systems at the architecture-level. Usually, a graph expresses a thousand of words. However, if a UML diagram containing too many of graphical elements such as rectangle, lines or arrows, describes too details of the software architecture, then the design document may go beyond the normal person's cognition. Such a software architecture may prevent from discovering design faults hidden in the overwhelmingly complicated diagrams.

Definition 2.1: VE of a UML diagram (an architecture view) is defined as the sum of each modeling element's area.

A great value of VE means that much space is needed in order to clearly place a great number of graphical modeling elements in a diagram. Consequently, the software system is complex. The area of a graphical element is computed analogously to the geometric formula "height×width". However, simplifying formulas to calculate VEs can be derived for different UML diagrams.

In the structure diagram, classes and interfaces modeled as rectangles have greater impact on human eyes than associations modeled as lines or arrows. The value EV of a class or interface varies according to the number of stereotypes and operations defined in them: the more stereotypes and operations a class has, the more space the class may occupy. Consequently, the formula to calculate EV for a class diagram is defined as following:

$$VE(class_diagram) = \sum class + \sum interface + c1 \times \sum relation + c2 \times \sum operation + c3 \times \sum attribute + c4 \times \sum stereotype \quad (2.1)$$

$0 \leq c1, c2, c3, c4 < 1$

It is assumed that the influences of relations, operations, attributes and other stereotypes have less impact than classes and interfaces. The four coefficients may be derived from the hierarchy of UML elements according to Fig. 2. For instance, if the all coefficients are assigned to 0.25, the value of VE in Fig.1 equals to $6+0.25 \times (10+20)=13.5$.

In a sequence diagram, actors and objects are arranged side by side above their interactions. This layout determines the width of a sequence diagram. The interactions among actors and objects modeled as arrows are laid vertically top down, which determine the height of a sequence diagram. Supposed that the height of actors and objects is defined as 1, and the

height of messages and responds is halved, then the VE of sequence diagrams is calculated as follows:

$$VE(sequence_diagram) = (\sum actor + \sum object) \times (1 + 0.5 \times (\sum message + \sum respond)) \quad (2.2)$$

Similarly, simplified EV computation can be derived for other UML diagrams.

Definition 2: VE of an architecture with different views described in UML diagrams is defined as follows.

$$VE(architecture) = \sum_{diagram \in architecture} \frac{VE(diagram)}{count_of_diagram(architecture)} \quad (2)$$

3.3. Connectivity degree

The measure CD expresses the extent to which entities in a UML diagram connect one another. Generally speaking, the goals of a system architect are to design the software architecture possibly with a clear overall structure, well-defined interfaces, and independency of components.

For a class diagram, this metrics is chiefly dependent on the relations among classes and their relations to interfaces. The more of relations exist in a class diagram, the more complex the diagram describes the software system, and the more difficult to understand the system. A small diagram with a lot of relations between classes and interfaces is more complicated than a large diagram with a few relations. The simplest class diagram consists of n classes and n-1 relations.

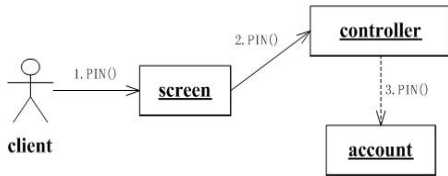


Figure 3 A simple communication diagram

Similarly, the interactions between objects and actors determine the CD value in a communication diagram. Figure 3 shows a simple communication diagram composing three objects, one actor and three messages. If the number of objects keeps unchanged, the complexity of the communication diagrams increases as more messages are exchanged between objects. The diagram will be more difficult to understand.

From the above discussions, we derive the following definition to calculate Connectivity Degree of a UML diagram.

Definition 3.1: CD of a UML diagram (an architecture view) is the average number of associations per entity.

In order to normalize the CD value to one for the simplest type of diagrams with n entities and n-1 associations, we decrease the number of entities by one, and define the CD value as one if there is only one entity in the diagram. Hence, we have the following formula for CD.

$$CD(diagram) = \begin{cases} 1, & \text{there is only one entity} \\ \frac{\sum association}{(\sum entity - 1)}, & \text{others} \end{cases} \quad (3.1)$$

In software architecture, components and connectors stand for entities, the bindings of ports or roles for associations. Obviously, a diagram without any entities is meaningless. According to formula (3.1) the CD of the diagram in Fig. 1 is 2.5, the CD of the diagram in Fig. 3 is 1, which is the simplest type of UML diagrams.

When taking the different types of association into consideration, a weight may be assigned to each association. The adjusted CD is computed as follows:

$$CD(diagram) = \begin{cases} 1, & \text{only one entity} \\ \frac{\sum_{i=1}^n association_i \times weight(association_i)}{(\sum entity - 1)}, & \text{others} \end{cases} \quad (3.2)$$

The above formulae are applicable to other UML diagrams, e.g. sequence diagram, activity diagram, and state diagram.

Definition 3: VE of an architecture with different views described in UML diagrams is defined as follows.

$$CD(architecture) = \sum_{diagram \in architecture} \frac{CD(diagram)}{count_of_diagram(architecture)} \quad (3)$$

4. Application of the metrics in architecture evaluation

The proposed metrics can be used to evaluate the internal quality attributes of architecture artifacts directly from UML diagrams, and to reveal external characteristics of software such as modifiability, maintainability, and stability. This section discusses the usefulness of these metrics in quantitatively evaluating software architecture.

4.1. Theoretic analyses

4.1.1. Architecture Scale. The scale of a software system is determined chiefly by its number of

composing elements: the greater the number is, the larger and more complicated is the system. The architecture scale defines the quality properties. The measure IC quantitatively describes the scale of software architecture and can be applied to evaluating this property. The great value of IC shows clearly that the architecture is composed of a large number components, connectors or interactions between them. Thus, the software system is likely complicated. Different kinds of constituents in the software architecture have different effects on the system. A component plays more important role in software architecture than a connection point that prescribes a point at which a component can join a connector to communicate with other components. The different roles of various modeling elements in software architecture are considered in the definition of IC. IC counts up not only each modeling element in the architecture, but also differentiates their impacts on the software quality.

Other UML measurements also take account of classes, interfaces, operations, attributes or stereotypes in UML diagrams. However, they simply summarize the number of the modeling elements without considering their different impact on the software system. For example, the scale of a system described in [16] is just defined as the total number of classes, their attributes and operations, and relationships between classes.

4.1.2. Architecture complexity. System complexity refers to the abstract representation of the effort and the time to develop it, as well as number of faults hidden in the final system. At the architecture level, the complexity may mainly imply the system's quality attributes like understandability, analyzability, and changeability.

Firstly, concise and well-structured diagrams are expected, since they make the software architecture easy to understand and modify. The measure VE is used to determine the optical effect of diagrams on human beings. VE contains the internal semantics of each modeling element as well as its external form, and is a suitable indicator for the complexity of graphically described software architecture. Apparently, a too large or too small VE value implies some quality faults for a software system. A great VE value means a lot of graphical elements drawn within a limited area. Such a diagram is confusing and is not suitable as architecture documentation. On the other hand, a small value may indicate compact and clearness of the design in a diagram. However, understanding the entire system may become more difficult, since the components and their interactions may be scattered in many different diagrams.

The visual representation belongs to one of the specific characteristics of graphical ADLs including UML. To our best knowledge, few quantitative papers on visual expression of UML notations have been published, if any.

Secondly, the complexity of a system depends chiefly on the communication and cooperation among its constituents. These relationships can cause much effort from a seemingly simple change. For instance, if the function of a component need to be modified, other components in the system that are connected with it have to be analyzed or even changed. Sometimes the entire system might be involved in the changes. The measure CD uniformly indicates the connection complexity of different diagrams that model different aspects of a system at different levels of abstraction. Particularly, the CD of sequence diagrams that describe the interactions between components indicates the system's global behavior.

4.1.3. Architecture stability. The system stability refers to the capability of the system structure and its components to avoid the risk of unexpected effect of modifications, which include changes of the system configuration, interfaces or implementation of components, or even replace of some components. At the architecture level, this quality can be expressed with two sub-characteristics, namely coupling and modularity (cohesion). The coupling of the architecture is a global property relative to the exchanges between two components. The cohesion expresses the logical topology of the architecture, as the number of components depending on other components.

The metrics CD can be directly used to measure the coupling and modularity of the architecture. But the definition of CD does not consider the directions of interactions between two components and the range of a conceptual module. Let us see an example. If component X requires a service from component Y in the architecture, then there exists a dependency of component X on Y. In this situation, the service provider Y does not care whether component X or any others need its service. Component Y exists absolutely independent of component X. But component X can not exist alone. However, if component X and Y are nested in another component Z, then the dependency of X on Y or the communication between them have surely impact on the functionality of component Z. Taking these discussions into account, we have the following more concise definitions to measure the coupling and modularity of an architecture.

Definition 4: COH of an architecture with different views described in UML diagrams is defined as follows.

$$COH(architecture) = \frac{\sum_{diagram \in architecture} COH(diagram)}{\sum_{diagram \in architecture} count_of_diagram(architecture)} \quad (4)$$

Where $COH(diagram) = \sum_{E \in diagram} COH(E)$ is COE of a diagram, $COH(E) = CD(E) / \sum fan_in(E)$.

Definition 5: COP of an architecture with different views described in UML diagrams is defined as follows.

$$COP(architecture) = \frac{\sum_{diagram \in architecture} COP(diagram)}{\sum_{diagram \in architecture} count_of_diagram(architecture)} \quad (5)$$

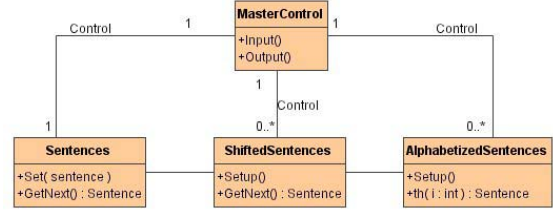
Where $COP(diagram) = \sum_{A, B \in diagram} COP(A, B)$ is COP of a diagram,

$$COP(A, B) = \begin{cases} 0, & \text{there is no relation(A,B)} \\ \frac{CD(A)}{\sum relation(A, B)}, & \text{there is relation(A,B)} \end{cases}$$

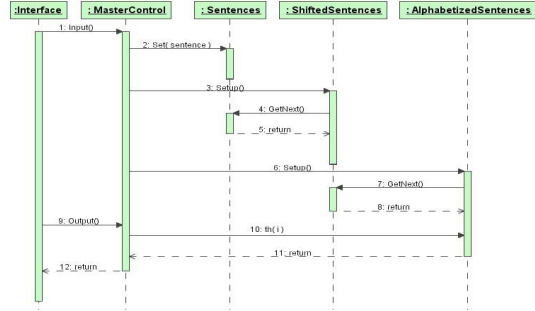
The metrics COH and COP indicate to what extend that an architecture component depends on other components or the encompassing system in order to fulfill its functions. For example of Figure 1, component IS2000 contains several components and connectors communicated each other. Left ports represent services required by the component, right ports express services the component provides, and the ports at the bottom stand for the connection with the network. According to the above definitions, the cohesion of component IS2000 is 0.8, and its coupling to the network is 2.5.

4.2. Case study

In order to evaluate the UML measurements for software architecture, we made a case study of a well known problem. We restructured two architectural designs of KWIC with UML, calculated the metrics to examine if similar results with those of previously published could be reproduced [17]. The selected two styles of software architecture for KWIC are Abstract Data Type (ADT) and Shared Data solution. For each kind of architectural designs we used a class diagram to describe the static structure and a sequence diagram to model the dynamic behavior, which are shown in Fig. 4 and 5, respectively.

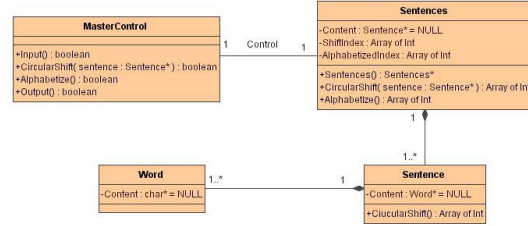


4 (a) Class diagram

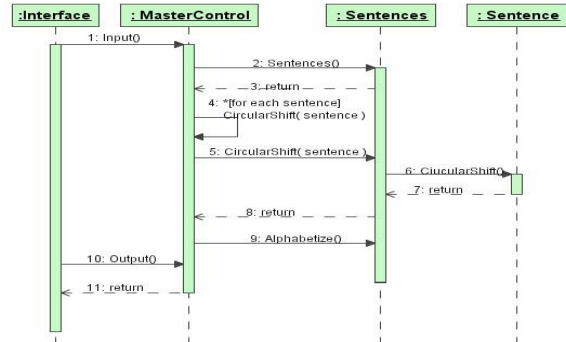


4 (b) Sequence diagram

Figure 4 KWIC architecture of ADT style



4 (a) Class diagram



4 (b) Sequence diagram

Figure 5 KWIC architecture of Shared Data style

Table 1 shows the three basic metrics for the two architectural designs. There are obvious differences of the measuring values between the two designs with respect to the same kind of UML diagrams such as sequence diagram. These demonstrate that the proposed metrics can differentiate quality attributes of

architectural designs with UML. The similar measurements of the whole architecture for each kind of the basic metrics indicate that class diagrams and sequence diagrams cannot be simply “added” together and compared. There are no average quality attributes of software systems at architecture-level. Architectural designs which are modeled from different views should be evaluated from different view-points with

Table 1 Metrics for different architectural designs of KWIC

	ADT architectural style		
	Class diagram	Sequence diagram	architecture
IC	18.00	15.00	16.50
VE	10.50	35.00	22.75
CD	1.67	3.00	2.33
	Shared Data architectural style		
	Class diagram	Sequence diagram	architecture
IC	19.75	13.25	16.50
VE	12.00	30.00	21.00
CD	1.00	3.67	2.33

respect to certain quality attributes.

5. Concluding remarks

This article presents three basic types of metrics IC, VE and CD based on UML. Their application to quantitatively evaluating quality characteristics of software architecture is discussed and showed with a case study. In order to use the measurements in the practice, the following works are necessary:

- (1) Studying the effects of modeling elements at different levels in the hierarchy of UML diagrams on the metrics, and devising a method to determine the weights and coefficients in the metrics definitions.
- (2) Finding the quantitative relationships of the metrics to certain quality attributes of software architecture with more controlled experiments.
- (3) Investigating hybrid evaluating methods by combining scenarios with quantitative measurements.

Acknowledgements: The authors thank the anonymous reviewers for their constructive suggestions on the paper.

6. References

- [1] L. Dobrica, and E. Niemela, “A survey on software architecture analysis methods”, *IEEE Trans. on Software Eng.*, 2002, 28(7), pp.638-653.
- [2] P. Clements, R. Kazman, M. Klein, *Evaluating software architecture*, Addison-Wesley, Boston, 2002.
- [3] M. Shaw, P. Clements, “The golden age of software architecture”, *IEEE Software*, 2006, 23(2), pp.31-39.

- [4] J. Rumbaugh, I. Jacobson, G. Booch, *The Unified modeling language reference manual*, Addison-Wesley, Bonn, 1999.
- [5] P.B. Kruchten, “The 4+1 view model of architecture”, *IEEE Software*, 1995, 11(6), pp.42-50.
- [6] C. Hofmeister, R. Nord, D. Soni, *Applied software architecture*, Addison-Wesley, Reading MA, 2000.
- [7] N. Medvidovic, J. Taylor, “A classification and comparison framework for software architecture description languages”, *IEEE Trans. on Software Eng.*, 2000, 26(1), pp.70-92
- [8] N. Medvidovic, D. Rosenblum, D.F. Fedmiles, J.Robbins, “Modeling Software architecture in the unified modeling language”, *ACM Trans. on Software Eng. and Methodology*, Jan. 2002, 11(1), pp. 2-57.
- [9] C.F.J. Lange, M.R.V. Chaudron, J. Muskens, “UML software architecture and design”, *IEEE Software*, 2006, 23(2), pp. 40-46.
- [10] G. Chastek, R. Ferguson, Toward Measures for Software Architecture. Technical Note, CMU/SEI-2006-TR-013, Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, March 2006.
- [11] M. Shereshevsky, H. Ammari, N. Gradetsky, “Information theoretic metrics for software architectures”, In *Proc. of 25th annual international computer software and applications conference*. Chicago, Illinois, 2001, pp. 151-160.
- [12] S. Yacoub, H.H. Ammar, T.A. Robinson, “Methodology of architecture-level risk assessment using dynamic metrics”, In *Proc. of the 11th International symposium on software reliability engineering*, San Jose, CA, 2000, pp. 210-221.
- [13] N. Subramanian, L. Chung, “Metrics for Software Adaptability”, In *Proceedings of the Software Quality Management Conference*, April 18-20, 2001, Loughborough, UK, pp. 98-108.
- [14] R. Bahsoon, W. Emmerich, “Evaluating the Stability of Software Architectures with Real Options Theory”, In *Proc. of the 20th International Conference on Software Maintenance (ICSM 2004)*, Chicago Illinois, USA, 2004.
- [15] P.O. Bengtsson, Architecture-Level Modifiability Analysis. Dissertation Series No. 02/02, Blekinge Institute of Technology, 2002.
- [16] L. Nenonen, J. Gustafsson, J. Paakki, I. Verkamo, Measuring object-oriented software architecture from UML diagrams. Technical Report, Department of Computer Science, University of Helsinki, Finland, 2000. <http://www.cs.helsinki.fi/index.html>
- [17] M. Shaw, D. Garlan, *Software architecture – Perspectives on an Emerging Discipline*, Prentice Hall, Inc. 1996, pp. 33-38.