HCTL Open International Journal of Technology Innovations and Research (IJTIR)
http://ijtir.hctl.org
Volume 12, December 2014
e-ISSN: 2321-1814, ISBN (Print): 978-1-62951-791-9

An Algorithmic Approach for Clone Clustering using Software Matrices and SOFM's

Himanshu Chaudhary and Ramesh Belwal

himx333@gmail.com

Abstract

With the ever increasing number of software applications and the critical risks which arise due to the low quality software, the utter importance of high quality software development gets to be more important than ever. The model that we have defined and illustrated here tries to provide an explicit process for ultimately binding quality-carrying properties into the desired software. These properties simply imply particular quality attributes in turn. In our research work, we proposed a quantitative Clone Detection model with respect to the Component Based Development (CBD) methodology using SOFMs. We have used the application of C. K. matrices so as to find out the cloning structure from each and every class of various types of design patterns (components). While we add up the matrices value for each and every component, the number of classes is also calculated at exact run time and concated with the final matrices value. This in total acts as an input for our neural network. Then neural network is used to train the self organizing map (SOM) and then the neural networks are used to find out in which case, maximum performance can be achieved. By using both the examples of design patterns and unsupervised neural network, we have proposed a dynamic model that provides far better performance for software component quality model.

Keywords

Clone Detection, Clone clustering, Software Matrices, Cohesion and Coupling, SOM, SOFM

Introduction

Software products have very significantly changed the facets of the work we do and also our daily life. Software products can be seen everywhere from the working place, offices, cars, the Internet, etc. Also, almost each and every electronic equipment contains some kind of software.

[1]M-Tech Scholar, BTKIT Dwarahat, Uttarakhand, India
[2]Assistant Professor, BTKIT Dwarahat, Uttarakhand, India

Softwares help in supporting and also running manufacturing industry, education, entertainment, health and financial services, economic analysis, research, management activities, and various other domains. Software development has become one of the largest industries in the world. When we talk of software engineering as a discipline, it emerged in the late 1960s and became increasingly important during the past 40 or 50 years. The overarching objective of basic software engineering is to bring out sound engineering disciplines for the development of software products so as to improve their overall quality. However, unlike other engineering disciplines, software engineering can still be considered a very young discipline, yet it has its extraordinary characteristics. It's inherently complicated, multi-disciplinary and multi-dimensional nature make it a very unique discipline which requires some specific solutions through combining knowledge from various disciplines.

Creation of a reliable, maintainable, etc. software system that satisfies the customer requirements is an inherently hard task, which is slowly and steadily becoming more and more difficult. Most importantly the size and complexity of the software systems are growing almost constantly. In many of the cases, small teams of developers can not for very long deliver newer product versions on the mentioned schedule because of the ever increasing number of features. As the overall number of developers grows, the management of development team and also the systematic and efficient communication flow and document sharing among the involved members becomes truly essential. Most software development organizations now much routinely use third party libraries, components and services in their own projects. They at most of the times also experience pressure from the customers to deliver customized and easily maintainable systems very quickly and cheaply. Software Engineering studies the common methods and techniques that help develop a very reliable, efficient, maintainable and evolvable software system on time and in budget, thus alleviating difficulties that are discussed above.

There are many definitions of Software Engineering. The definition given in IEEE states that: “Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software” [1].

Software Module Clustering

Software module clustering has been the most important as well as challenging problem in the field of software engineering. It is very widely believed that a well-modularized software system will be very easy to develop and maintain. Generally, a good module structure is regarded as the one that has high degree of cohesion and also low degree of coupling. Sadly, as softwares evolve, modular structures tend to degrade necessitating a process of restructuring so as to regain the cognitive coherence of the previous incarnations. We are concerned with automated techniques to suggest software clustering’s, delimiting boundaries between modules so as to maximize cohesion while minimizing the coupling.

Many authors have very widely considered the implications of software modularization on many software engineering concerns. Badly modularized software is very widely regarded as a common source of problems for comprehension, also increasing the time of ongoing maintenance and testing. The use of cohesion and coupling for assessing modular structure was first popularized by work of Constantine and Yourdon, who introduced the seven-point scale of cohesion and coupling measurement. Such level of cohesion and their measurement has formed the topic of much sought after work so as to define software metrics to compute them and also to assess their impact on development of software.

There exist many ways of approaching the module clustering problem. Following Mancoridis et al., who firstly suggested the search-based approach to module clustering we follow the search-based approach. In this search based approach, the attributes of good modular decomposition are generally formulated as objectives, the evaluation of which as a "fitness function" will guide a search-based optimization algorithm.

Related Work and Approaches

Code duplication, or cloning, is a common programming practice that can lead to serious maintenance issues if not properly managed. The solution to managing clones has traditionally focused on clone detection and subsequent removal. Unfortunately, detecting clones is a difficult problem. Two identical or similar code fragments form a clone pair. Previous studies have identified cloning as a risky practice. Therefore, a developer needs to be aware of any clone pairs so as to properly propagate any changes between clones. A clone pair experiences many changes during the creation and maintenance of software systems. A change can either maintain or remove the similarity between clones in a clone pair. If a change maintains the similarity between clones, the clone pair is left in a consistent state. However, if a change makes the clones no longer similar, the clone pair is left in an inconsistent state. The set of states and changes experienced by clone pairs over time form an evolution history known as a clone genealogy. In this thesis, we provide a formal definition of clone genealogies, and perform two case studies to examine clone genealogies.

In literature many authors have very considered the implication of software modularization on Software Cloning. Badly modularized software is very widely regarded as a common source of Cloning; Clones also increase the time of ongoing maintenance and testing. We propose the use of cohesion and coupling under C. K. Matrices for detecting Clones in the Software Projects. Furthermore our approaches allow us the utilities by which we can visualize the clones effectively. In our approach we will utilize the Kohonen Self Organizing Map for clustering clones in component based Systems. Self-Organizing Maps and networks are important tools for pattern recognition, clustering, speech recognition, data compression, medical diagnosis.. The benefit of SOFMs is that they are unsupervised learning models, i.e. for detecting clones we do not need to specify existing clones they cluster the clones by themselves. However, the results obtained by the Kohonen networks are dependent on their parameters such as the architecture of the Kohonen map, the later has a great impact on the convergence of learning methods.

The Proposed Methodology

The Main Objectives of our research are the implications of the SOFMs in detecting coupling between component models based on C. K Matrices which in turn points to Clones in the given Software models, the research goal of the project are as follows:

1. Firstly to learn and Understand the Self Organizing Map and they working in Matlab.
2. Secondly to collect Design patterns and C. K Matrics for various software products which have high coupling and cohesion i.e chances of detecting clones is higher.
3. To create Layer of ANN based SOFM for Learning Patterns in input C. K matrics
4. The Matrices will then be transferred to learning layers of SOFMs, The SOFMs will be responsible for detecting clones.
5. The Clustered clones will be visualized by the plotting utilities with the help of SOM network.

The Proposed Algorithm

For the effective visualization of the Self-Organizing Feature Map (SOFM) is the most efficient as well as effective tool. In its purest and most basic form it can produce similarity graphs of the data that has been provided as the input. These generally convert the non-linear statistical relationship amongst the high-dimensional data to simple geometric relationships onto the display which is low- dimensional, in most of the cases it is a regular 2-D grid of nodes. The SOFM on one hand compresses the present information alongside the preservation of the most important topological, metric relationships of present primary data element on the featured display, this thing can also be thought to do the production of some kind of abstractions also. Visualization and abstraction are the two important aspects, and can be very effectively used in a number of ways for solution of complex tasks which include analyzing a process, machine perception, recognition of speech, quantization of a vector, equalization of adaptive nature and last but not the least combinational optimization.

SOFM is a kind of model that is much more of general nature as compared to the ANNs model. The former is absolutely capable of creating or generating mappings from high dimensional signal spaces onto the lower dimensional topological structure .These are generally performed adaptively in a fashion that is topologically ordered. These mappings tend to create topologically neighborhood relationships geometrically explicit in the low dimensional feature maps.

Algorithm for Detecting Clones Using the SOFMs

The neural network training of the SOFM C. K matrices as the feature maps is as under:

1. Initialization of the SOM: First of all we need to select or choose some random values for the initial weights which is Wj (0).

2. Now we need to set the size of neurons into the SOM ***MxN***

3. We will be having input matrices which will be commonly selected from the input space and they will be denoted by

   $\mathbf{X} = [x_1, x_2, \ldots, x_m]$

   We need to note that m is dimension of given input data space. For any neuron j, it's weight vector will be denoted as:

   $\mathbf{W_j} = [w_{j1}, w_{j2}, \ldots, w_{jm}]$, j = 1, 2, 3, 4, 5, 6, 7, 8, 9 …… N

   Here N is to be considered as the total number of the neurons in SOM layers.

4. I_{mxn} is defined as a matrix of Patterns in which Code Matrices occur, and which has been derived from the Software Components

   Also here, M can be defined as the total number of rows.

   The number of columns existing in *I* is n.

   The first and the foremost task of SOM based clone code detectors is the task of finding the main cluster centers for every input x which is contained in the input space X,

5. Do, for each and every of the C.K matrices which are from the software components along with features n in number, add a row ***r*** in ***I***

   Now we intend to find out the neuron for each ***i*** so as the weight vector for C.K matrices is optimally the closest to the input vector, which means that;

   thereafter i(x) will be determined as under:

   $\boldsymbol{i}(\mathbf{x}) = \arg\min \|X - W_j\|$, j = 1,2,3,4,5,6,7,8,9…,L

6. Now comes the turn for updating the weights w_i , i = 1,2,3,4,5,6,7,8,9…m

   $w_i(t + 1) = w_i(t) + \eta(t) * X(i, i^*t)\,(x_i - w_i(t))$, η is the learning rate for the SOM.

7. Until then, try changing the Sum of weights which has to be certainly greater than the threshold $\sqrt{\sum_{i=1}^{m} |\Delta w_{ij}|^2} > \epsilon$

   ∊ is the maximum threshold

8. wt = weighted sum has to be calculated for the neurons present in the SOM layers.

9. wt weights will now be in a need to be sorted in a decreasing order.

The weighted sums are to be plotted and the neuron structure is finally exited.

Results and Analysis

The complexity of the internal structure of a particular component can help estimate the effort related to the evolution of that particular component. In our ongoing research, we focus on quality of internal design of the software component as well as its direct relationship to the external quality attributes of the desired component. The complete work has been classified into the following phases.

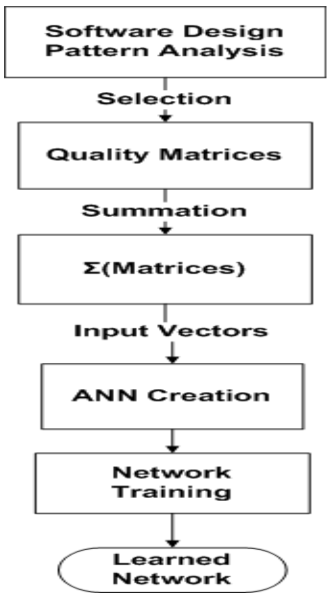


Figure 1: System Flow chart for SOM training for Detecting Clones

- **Analysis Phase**: Calculating different matrices like WMC (Weighted Methods per Class), DIT (Depth of Inheritance Tree), NOC (No. of Children), RFC (Response For a Class), CBO (Coupling Between Objects). These are computed for every class of design patterns.

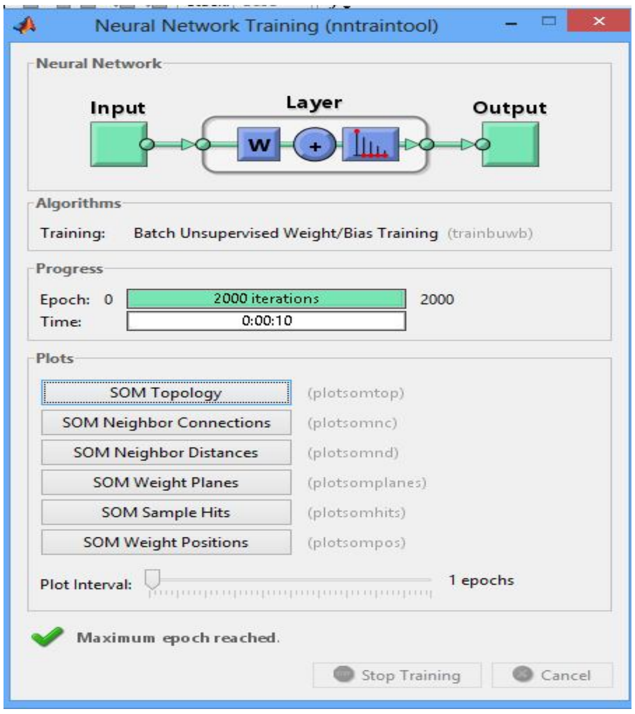
- **Summation** of matrices for every class is done for each and every design pattern. Also another matrix NC (No. of classes is used in each pattern) and is calculated at the run time.

- **Network Creation** An unsupervised neural Network is particularly created for this Classification Problem.

The steps that are being followed to obtain an improvement in the performance are followed as stated earlier. Number of training data is 21 x 6 i.e. 126 elements. The results of the experiment for training of neural networks in the MatLab are shown in table 4. In the table, the four different values used are training method, number of training data, number of epoch taken to converge, number of output data, and the time taken for the execution of program are shown.

Experiment	**Experiment**
Training method used	Trainbuwb
No. of training data	6 x 21 = 126
No. of epoch taken to converge	2000
Time taken to execute	6.40535 sec.
No. of outputs	6

Table 1: The experimental results using neural network analysis



Training: Batch Unsupervised Weight/Bias Training (trainbuwb)

Epoch: 0 2000 iterations 2000
Time: 0:00:10

Plot Interval: 1 epochs

Maximum epoch reached.

Figure 2: Neural Network Training using ***nntraintool***

Weight positions as generated by the SOM. According to figure, we take input value in neural training network by using nntraintool. Unsupervised weight Batch are used in the algorithm. This does 2000 iterations in 10 seconds. There are tasks that are much more suited to an algorithmic approach like arithmetic operations and also tasks that are better suited to neural networks. Even more, a very large number of tasks require system that uses a perfect combination of the two defined approaches so as in order to perform at the maximum efficiency.

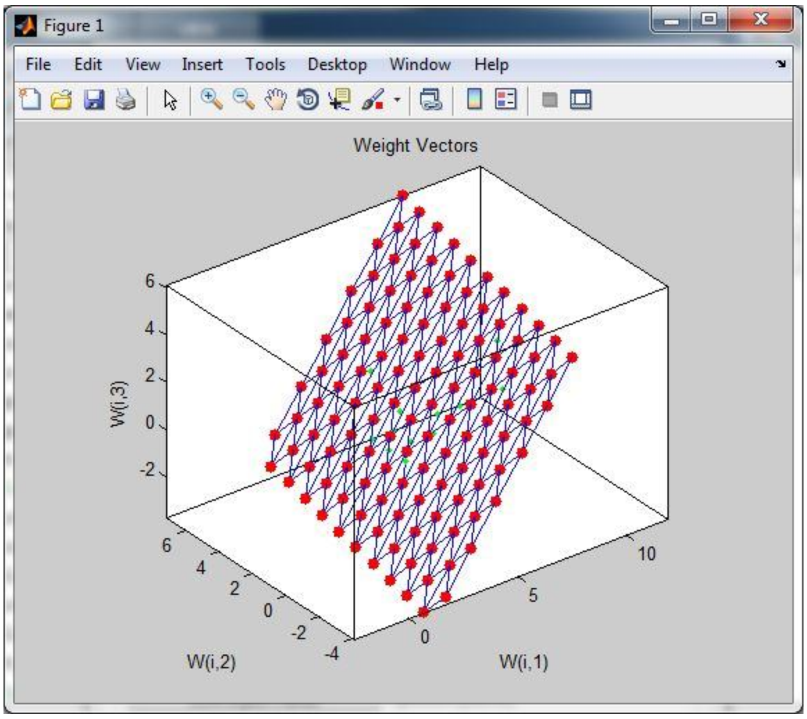


Figure 3: SOM weight positions before training

According to figure, it is aligned towards the diagonal. As per the results which are shown above, the model proposed for the design pattern performance can be improved by having weighted method per class as 6, depth of inheritance tree as 6, response for class as 9, number of children as 6, number of classes as 6 and coupling between objects as 4. The total time elapsed in execution of the proposed model takes only approximately 6.4 seconds.

Adding up the matrices value for each and every single component, number of class is also calculated at the run time and concated with the final matrices value. This in total acts as the desired input for the neural network. Then the neural network is used to train the self organizing map (SOM) neural network and also to find out that in which case, maximum performance can be achieved. By using the example of design patterns and unsupervised neural network, we have proposed a specific model that generally provides far better performance for software component quality model, but from the above results.

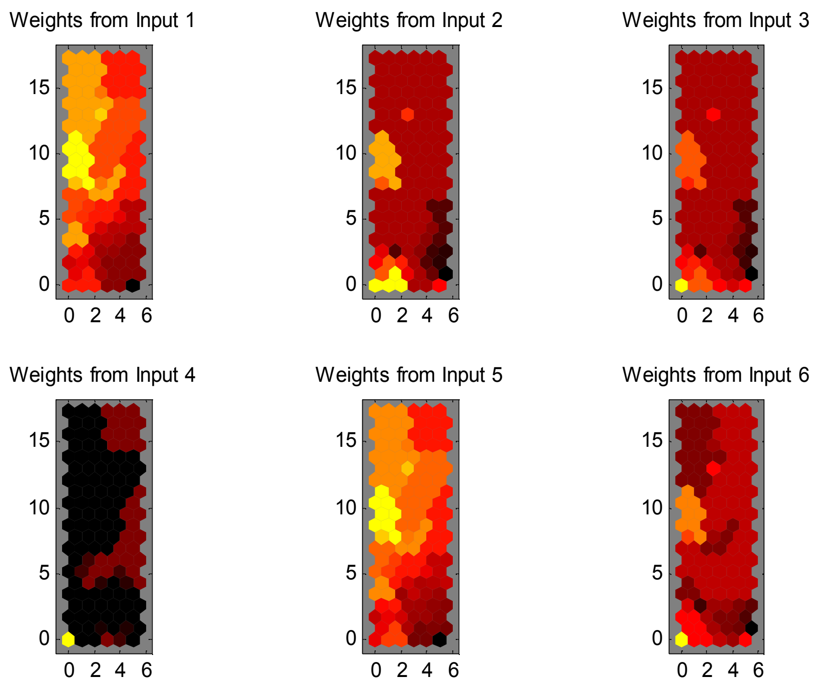


Figure 4: SOM weights for all inputs after Training phase.

The results shown using MatLab through execution of Neural Network are:

Weighted Method per Classs (WMC): 6

Depth of Inheritance Tree (DIT): 6

Response For Class (RFC): 9

Number Of Children (NOC): 6

Number of Classes (NC): 6

Coupling Between Objects (CBO): 4

Total time elapsed in entire execution: 7.14773 seconds.

Conclusions and Future Work

In our research work, we proposed a quantitative Clone Detection model with respect to the Component Based Development (CBD) methodology using SOFMs. We have used the application of C. K. matrices so as to find out the cloning structure from each and every class of various types of design patterns (components). While we add up the matrices value for each and every component, the number of classes is also calculated at exact run time and concated with the final matrices value. This in total acts as an input for our neural network. Then neural network is used to train the self organizing map (SOM) and then the neural networks are used to find out in which case, maximum performance can be achieved. By using both the examples of design patterns and unsupervised neural network, we have proposed a dynamic model that provides far more better performance for software component quality models.

From the above results, it can be inferred that performance can still be much more improved if more realistic data is available for acting as the training set and also if the neural network techniques along with fuzzy techniques are considered. Also, if the output may be known from some past values, a supervised neural network may give better results.

We also need to assess the effectiveness by applying different types of software products (e.g., embedded, commercial, and developed software) and also to compensate the metrics and their weights in a much more continuous manner. Moreover, we also require a quantitative software quality evaluation process by successfully elaborating our model into various phases, activities, and artifacts for full practical use. The full advantage of component-based clone clustering approach will be achieved when not just the C. K matrices, but also hybrid approaches with SOMs and iterative methods are used. This approach may ultimately lead to much easier and more accurate predictability of Clone Detection and visualization. In component-based approach, many more system attributes can easily be derived from the component attributes, and this will be much more accurate if a general support for defining and obtaining measurements of the attributes are built in the component technologies themselves and also if there are well defined restriction rules using these technologies.

References

[1] Abhikriti Narwal, "Empirical Evaluation of Metrics for Component Based Software Systems", International Journal of Latest Research in Science and Technology, Vol. 1, Issue 4, pp.373-378, Nov.- Dec. 2012.

[2] Amr Rekaby, Ayat Osama, "Introducing Integrated Component-Based Development Lifecycle and Model", International Journal of Software Engineering & Applications (IJSEA), Vol.3, No.6, pp. 87-99, Nov. 2012.

[3] Sandeep Srivastava, "Software metrics and Maintainability Relationship with CK Matrix", International Journal of Innovations in Engineering and Technology, Vol. 1 Issue 2, pp. 76-82, Aug. 2012.

[4] Simrandeep Singh Thapar, Paramjeet Singh, Shaveta Rani, “Challenges to the Development of Standard Software Quality Model”, International Journal of Computer Applications, Vol. 49, No.10, pp. 1-7, July 2012.

[5] Anupama Kaur, Himanshu Monga, Mnupreet Kaur, Parvinder S. Sandhu, “Identification and Performance Evaluation of Reusable Software Components Based Neural Network”, International Journal of Research in Engineering and Technology, Vol. 1, No. 2, pp. 100-104, March 2012.

[6] G. Shanmugasundaram, V. Prasanna Venkatesan, C. Punitha Devi, “Reusability metrics - An Evolution based Study on Object Oriented System, Component based System and Service Oriented System”, Journal Of Computing, Volume 3, Issue 9, pp. 30-38, Sept. 2011.

[7] Aldeida Aleti, Indika Meedeniya, “Component Deployment Optimisation with Bayesian Learning”, ACM Journal, pp. 11-20, June 2011.

[8] Samira Si-saïd Cherfi, Jacky Akoka, Isabelle Comyn-Wattiau, “Federating Information System Quality Frameworks Using A Common Ontology”, Proc. 16th International Conference on Information Quality, pp. 160- 173, 2011.

[9] Mostefai Mohammed Amine, Mohamed Ahmed-Nacer, “An Agile Methodology For Implementing Knowledge Management Systems : A Case Study In Component-Based Software Engineering”, International Journal of Software Engineering and Its Applications, Vol. 5, No. 4, pp. 159-170, 2011.

[10] Anju Shri, Parvinder S. Sandhu, Vikas Gupta, Sanyam Anand, “Prediction of Reusability of Object Oriented Software Systems using Clustering Approach”, World Academy of Science, Engineering and Technology, Vol. 43, pp. 853-856, 2010.

[11] V. Lakshmi Narasimhan, P. T. Parthasarathy, M. Das, “Evaluation of a Suite of Metrics for Component Based Software Engineering (CBSE)”, Issues in Informing Science and Information Technology, Vol. 6, pp. 731-740, 2009.

[12] María A. Reyes, Maryoly Ortega, María Pérez, Anna Grimán Luis E. Mendoza and Kenyer Domínguez, “Toward A Quality Model for CBSE”, International Conference on Enterprise Information Systems, pp. 101-106, 2009.

[13] R. Senthil, D. S. Kushwaha, A. K. Misra, “An Extended Component Model and its evaluation for Reliability & Quality”, in Journal of Object Technology, vol. 7, no. 7, pp. 109-129, Sept. 2008.

[14] Yoonjung Choi, Sungwook Lee, Houp Song, Jingoo Park, SunHee Kim, “Practical S/W Component Quality Evaluation Model”, ICACT, pp. 259-264, Feb. 2008.

[15] Mubarak Mohammad, Vasu Alagar, “A Component-Based Software Engineering Approach for Developing Trustworthy Systems”, ACTS Report Series, Feb. 2008.

[16] Anita Gupta, Reidar Conradi, Forrest Shull, Daniela Cruzes, “Experience Report on the Effect of Software Development Characteristics on Change Distribution”, Springer Journal, pp. 158–173, 2008.

[17] Kung-Kiu Lau, Zheng Wang, “Software Component Models”, IEEE Transactions On Software Engineering, Vol. 33, No. 10, pp. 709-724, Oct. 2007.

[18] Net Objective, “Design Patterns: From Analysis to Implementation”, Manuals for design patterns explained: A New perspective for Object Oriented Design, 2007.

[19] Alexandre Alvaro, Eduardo Santana de Almeida, Silvio Lemos Meira, “A Software Component Quality Model: A Preliminary Evaluation”, IEEE Proc. of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, 2006.

[20] Kilsup Lee, Sung Jong Lee, “A Quantitative Software Quality Evaluation Model for the Artifacts of Component Based Development”, Proc. of the Sixth IEEE International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005.