RESEARCH ARTICLE OPEN ACCESS

Quality Model For Analysis And Implentation Of CK Metrics Through Neural Networks

*Ashish Oberoi, **Deepti Arora
*Deptt. of Computer Science & Engg.,MMEC, Mullana
**Deptt. of Computer Application, Panipat Institute of Engg. & Technology, Samalkha

ABSTRACT
Component engineering addresses the issues of component's specification, development, qualification, documentation, cataloguing, and adaptation and selection for reuse. In general, software systems implement functional and non-functional requirements. This implies that component specification methods and qualification techniques should support both functional and non-functional requirements. The paper proposed a model for analyzing CK metric values of component-based software by systematically analyzing a series of metrics using CK metrics analysis and several key inferences are drawn from them. Software component design patterns are used for analyzing various metrics and drawing a number of useful conclusions by evaluating them, which will include inferences on reusability of the underlying components. By using a Self Organizing Map (SOM), empirical evaluation of CK metric component models is done that figure out various matrices which affects the performance of Component based Software Engineering Model and made a try to propose a model that by selecting what metrics of component model gives optimized metric values.

Keywords: Component Based Software Engineering (CBSE); Component Based Development (CBD), Neural Network(NN).

I. INTRODUCTION

Software reuse has long been one of the major issues in the world of software engineering. The reason is obvious. Software reuse can dramatically increase the productivity of the software community, ease maintenance, and improve product reliability. Although most people would agree upon the importance of reuse, it is only today that it has become a main goal in software engineering. As a result, many software reuse technologies have been developed over the past few years. A popular reuse technique in the object-oriented programming community is design patterns. Design patterns represent a recurring solution to a software development problem within a particular context. They have frequently been used to guide the creation of abstractions in the software design phase, necessary to accommodate future changes and yet maintain architectural integrity. These abstractions help us de-couple the major components of the system so that each component may vary independently.

Component-Based Software Engineering (CBSE) is a systematic and structured approach that allows software engineers to maximize reusability.

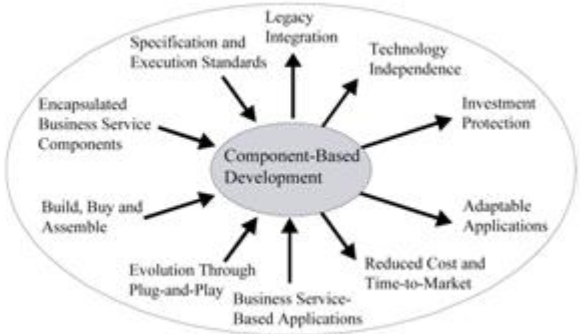

Legacy Integration
Specification and Execution Standards
Technology Independence
Encapsulated Business Service Components
Investment Protection
Component-Based Development
Build, Buy and Assemble
Adaptable Applications
Evolution Through Plug-and-Play
Business Service-Based Applications
Reduced Cost and Time-to-Market

Fig 1: Concept and Benefits of CBSD

CBSE is also known as Component-based Software Development (CBSD) concept and benefits of CBSD are shown in Fig 1.

In principle, CBSD should provide a software organization with advantages in higher productivity, reduced time to market; reduce the cost of development and higher quality system. The ensuring of quality of a component based system is an important task because unlike tradition software systems, the quality of a component based system depends both on the quality of its components and the framework being used. The development process and the maturity of an organization also influence the quality of component based products. Hence, it is easy to perceive that the quality of its components, directly or indirectly, influence the quality of the final software. CBSE is an extension of object-oriented concepts such as encapsulation (information hiding), abstraction (what an element is and how it should be implemented), polymorphism (same operation behave differently on different elements), Inheritance (sharing of operation and attributes among elements based on hierarchal relationships). CBSE provides many advantages like:

- Component-based software development can increase the productivity of software developers. Component-based software is constructed by assembling existing reusable components. This process is much faster than writing an application from scratch.
- Component-based software development offers higher quality, more reliable software. The main reason is that reusable components have been tested and therefore their quality can be assured.
- Component technology can ease software maintenance. Component-based software means that a large software application can be made of many small components. A task for maintaining a large software application can be partitioned to many smaller and easier tasks for maintaining components.
- Component technology makes it easier to manage software development. Component partitioning enables parallel development, allowing several organizations to be involved in development of larger and more complex software.
- Because component technology implies some base set of standards for infrastructure service, a large application can depend on these standards thereby saving considerable time and effort.

The fundamental concepts on which CBSE is based are:

1.1 Component

The word “component” is used very broadly and often loosely throughout the software industries. Generically, a component is defined as a computational unit. Components can be things like clients and servers, databases, filters, and layers in a hierarchical system.

“A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party”. Main points of this definition indicate that component can be deployed independently and each component interacts with other component(s) by using interfaces [7]. Other definition of component is:

“A component is a coherent package of software that can be independently developed and delivered as a unit, and that offers interfaces by which it can be connected, unchanged, with other components to compose a larger system”. Component-based software development means building software by assembling or gluing components together. Fundamental characteristics of components are presented as:

- **Independent:** A component must be independent from its environment and is deployed without needs of other specific components.
- **Standardized:** In CBSE approach a component should follow deployment and composition rule.
- **Deployable:** For a component to be deployable, a component has to be self contained and must be able to perform as a stand-alone entity on some component platform that implements the component model. Usually a component is a binary component and cannot be compiled before its deployment.
- **Documented:** A component should be specified formally.
- **Composable:** A component communicates with others through its public interfaces. Also, it must provide external access to information about itself such as its methods and attributes.

1.2 Interface

“An interface of a component can be defined as a specification of its access point”. An interface is a set of functional properties which includes set of actions understandable by both interface provider (component) and user (other components or other software that interact with provider). Clients access the services that are provided by a component through access points. A component may have more than one access point, which contains different services provided by that component. Therefore, a component may have more than one interface. Since components are black box, their implementation detail is not accessible from outside [9].

II. LITERATURE REVIEW

A continuous research is continuing on Component Based Software Engineering model. A brief review of the work done in the past is elaborated here:

In [2], authors presented a survey of component-based development and reuse driven development life cycles. The proposed model contains all the needed activities towards a complete component-based development lifecycle. A comparison between ICBD, normal component-based development, and non-component based development is provided.

In [4], authors discussed the key challenges to the development of standard, complete and pervasive software quality models, solution to these challenges and their importance. They discussed key issues which need to be considered to develop a widely acceptable standard software quality model. Important issues which are posing obstacles to the development of standard quality models are discussed. They lay down the foundation for the development of a component quality model which is comprehensive and may be used to increase to reusability aspects of components.

In [5], authors proposed explores the five algorithms, Fletcher–Reeves Update Conjugate Gradient (FRUCG) algorithm, Polak–Ribiere Update Conjugate Gradient (PRUCG) algorithm, Powell-Beale Restarts Conjugate Gradient (PBRCG) algorithm, Scaled Conjugate Gradient (SCG) algorithm, Self- Organizing/network algorithms based Neural Network are experimented to develop the reusability evaluation model for function oriented software systems and the results are recorded in terms of Accuracy, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).

In [6], authors discussed a study of the reuse metrics of three systems i.e. object oriented systems, component based systems and service oriented systems is made and proposed a model to bring out the relationship between them. A template has been designed to study and record how the metrics are categorized and it forms the base for the evolution based model. An evolutionary based model is proposed which states the maturity level of reuse metrics and identifies the gaps to measure complete reusability for service oriented systems.

In [8], an author proposed a model and discusses the main constituents of ontology of quality federating all the aspects of Information System (IS) components quality (software, data, models, etc.). In order to operationalize the proposed ontology, an approach is described that allows using the ontology in order to achieve specific quality goals. QualOnto as a framework is used to link together the IS engineering process and the IS product, which could serve as a basis for statistical studies on the correlation between both process and product qualities.

In [9], authors presented a new methodology of Knowledge Management System (KMS) implementation in a CBSE-oriented organization. A case study of applying this methodology in an existing CBSE organization is also presented. The main objectives of methodology used are an early elimination of risks and misconceptions by ensuring short iterations, continuous integration and intensive customer collaboration. The proposed methodology requires less resources and budget than existing methodologies.

In [10], authors proposed an algorithm in which the inputs can be given to K-Means Clustering system in form of tuned values of the Object Oriented software component. A hybrid K-Means and Decision tree approach is used to predict the reusability value of object oriented software components based on the metric values. The developed reusability model produces high precision results.

In [12], authors made an analysis of the conceptual elements behind Component-Based Software Engineering (CBSE) and proposed a model that support its quality evaluation and integrates the product perspective, a view that includes components and Component-Based Software (CBS), as well as the process perspective, a view that represents the component and CBS development life cycle. Two findings are highlighted that are: 1) a close relationship exists between both identified perspectives: quality of a component directly influences CBS quality; 2) the component models are the backbone of these software systems.

In [13], authors described N-tier architecture as data access architecture in a component based application and is evaluated against the external and internal quality factors. This establishes that an enhanced component model (ECM) is a reliable model. This expresses how data access objects (DAO) in the DAO layer interacts with the business-tier and data source in achieving reliable, reusable, robust and scalable component model by implementing Data Adapter interface.

In [15], authors presented a CBSE approach that involves three contributions. The *first* contribution is a component model that defines the trust worthiness quality attributes as first class structural elements. *Second* contribution is a process model role. The *third* and final contribution is a development framework of comprehensive tool support.

In [17], authors made a survey and analysis of current component models. Based on the analysis, they are classified into a taxonomy based on commonly accepted parameters for Component Based Development. For each category in the taxonomy, its key characteristics are described and evaluated with respect to these parameters.

In [19], authors proposed a component quality model which describes consistent and well-defined characteristics, sub-characteristics, quality attributes and related metrics for the components evaluation. A preliminary evaluation to analyze the results of using the component quality model is also proposed.

III. Proposed Work

Hereby, we are going to propose a model for measuring quality component e.g. reusability of a Component based Software Engineering model through analyzing a series of design patterns which are worldwide accepted as the reuse design terminology for object oriented designing and hence component based designing. Analyzing the entire design pattern values obtained [31].
The object-oriented metrics proposed [19] and later refined by the same authors can be summarized as follows:

a. Weighted Methods per Class (WMC): This is a weighted sum of all the methods defined in a class.

b. Coupling Between Object classes (CBO): It is a count of the number of other classes to which a given class is coupled and, hence, denotes the dependency of one class on other classes in the design.

c. Depth of the Inheritance Tree (DIT): It is the length of the longest path from a given class to the root class in the inheritance hierarchy.

d. Number of Children (NOC): This is a count of the number of immediate child classes that have inherited from a given class.

e. Response for a Class (RFC): This is the count of the methods that can be potentially invoked in response to a message received by an object of a particular class.

f. Lack of Cohesion of Methods (LCOM): A count of the number of method-pairs whose similarity is zero minus the count of method pairs whose similarity is not zero.

Here we will use unsupervised method because we don't know in advance output values for the corresponding design patterns [21] [27]. MatLab will be used for the implementation purpose. Firstly, analysis of Software Design Patterns through CK metrics analysis is made. Then, Implementation of the proposed model is done through Neural Network using MatLab.

3.1 Software Design Pattern Analysis

Design patterns represent a recurring solution to a software development problem within a particular context. They have frequently been used to guide the creation of abstractions in the software design phase, necessary to accommodate future changes and yet maintain architectural integrity. These abstractions help us de-couple the major components of the system so that each component may vary independently. Here we are going to discuss software design patterns in detail. Consider the example of Abstract Factory design pattern as shown in Fig 2.

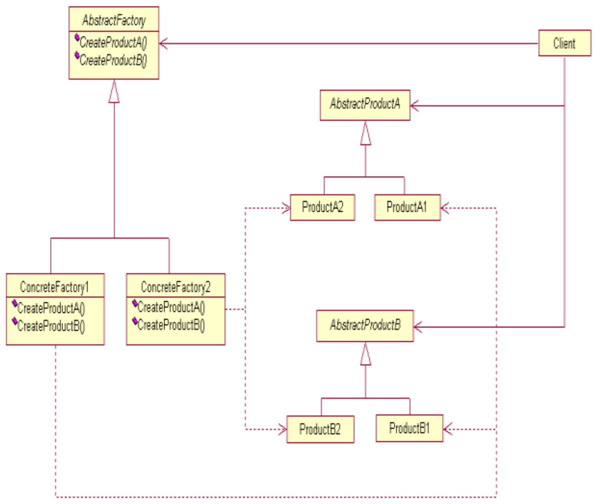


Fig 2: Abstract Factory Design

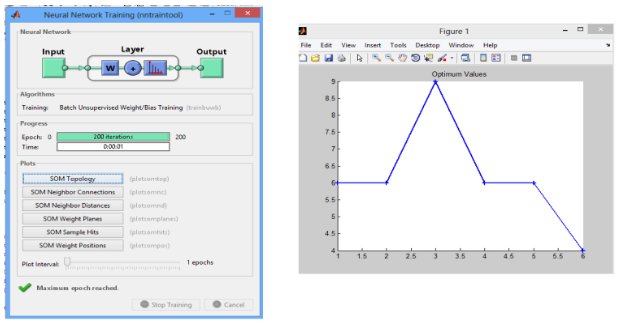
Class/Matrix	NOM	DIT	NOC	CBO	RFC
Abstract Factory	2	0	2	0	2
Concrete Factory 1	2	1	0	2	2
Concrete Factory 2	2	1	0	2	2
Abstract Product A	0	0	2	0	0
Product A2	0	1	0	0	0
Product A1	0	1	0	0	0
Abstract Product B	0	0	2	0	0
Product B2	0	1	0	0	0
Product B1	0	1	0	0	0

Table 1: CK Matrix Analysis for Abstract Factory Design

In the same way, CK matrix analysis for all the design patterns is done. Table 1 shows the ck matrix analysis for abstract factory design.

3.2 Neural Network Implementation

Hereby we have developed a Self Organizing Map Neural Network to train the network. Then summing up the computed weights w.r.t. every metrics and dividing the sum by total number of matrices which yields the optimized value among the matrices. Neural Network training and optimum value for the component based software engineering model is shown in Fig 3.


Neural Network Training (nntraintool)
Neural Network
Input
Layer
Output
W
Algorithms
Progress
Plots
SOM Topology (plotsomtop)
SOM Neighbor Connections (plotsomnc)
SOM Neighbor Distances (plotsomnd)
SOM Weight Planes (plotsomplanes)
SOM Sample Hits (plotsomhits)
SOM Weight Positions (plotsompos)
Stop Training
Cancel
Figure 1
File Edit View Insert Tools Desktop Window Help
Optimum Values
9
8.5
8
7.5
7
6.5
6
5.5
5
4.5
4
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6

Training: Batch Unsupervised Weight/Bias Training (trainbuwb)
Epoch: 0 200 iterations 200
Time: 0:00:01
Plot Interval: 1 epochs
Maximum epoch reached.

Fig 3: Neural Network Training and Value for the Proposed Model

IV. Result Analysis

The experiment results for training of neural network in the MatLab are shown in Table 2.

Table 2: Neural Network Analysis Values

Experiment	Values
No. of training data	5 x 21 = 105
No. of epoch taken to converge	200
Time taken to execute	2.94093 seconds
No. of outputs	5

The capability of neural network to generalize and insensitive to the missing data would be very beneficial. For training purpose all 21 design patterns and 6 matrices are taken. Number of epochs taken is 2000 to achieve high accuracy.

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

Performance can be improved by using:

Weighted Method per Classs (WMC):	6
Depth of Inheritance Tree (DIT):	6
Response for Class (RFC):	9
Number of Children (NOC):	6
Coupling Between Objects (CBO):	4

Total time elapsed in entire execution: 2.94093 seconds.

As per the results shown above, the model proposes for the design pattern performance can be improved by using the above values.

V. Conclusion & Future Scope

The model proposed and illustrated here provides an explicit process for adding quality-carrying properties into software. CBSE is a knowledge-intensive activity where collaborators produce and consume knowledge during all the development phases. CBSE is adding a lot of value to rapid application development and is actively contributing to better quality software systems.

In the paper, we proposed a model for enhancing quality with respect to the Component Based Software Engineering (CBSE) methodology. This in total acts as input for the neural network. By using the example of design patterns and unsupervised neural network, we have proposed a model that provides enhancing quality criteria for Software Component Engineering model based on CK metric values. If output may be known from the past values, a supervised neural network may give better results.

Component modeling techniques, with which we have compared our work, do not provide all the tools necessary for rigorous analysis at different stages of system lifecycle. The reason is that these component models are designed and implemented for different specific domains. In order to properly enable the evaluation of software components, supplying the real necessities of the software component markets, a component quality model is strictly necessary. A more elaboration on finding the appropriate size of software with respect to the overhead for preparation of evaluation is required. A full advantage of component-based approach can be achieved when not only the functional parts are reused, but also when this approach leads to easier and more accurate predictability of the system behavior.

References

[1] Abhikriti Narwal, “Empirical Evaluation of Metrics for Component Based Software Systems”, *International Journal of Latest Research in Science and Technology, 1(4),2012,373-378.*

[2] Amr Rekaby, Ayat Osama, “Introducing Integrated Component-Based Development Lifecycle and Model”, *International Journal of Software Engineering & Applications (IJSEA), 3(6),2012, 87-99.*

[3] Sandeep Srivastava, “Software metrics and Maintainability Relationship with CK Metrics”, *International Journal of Innovations in Engineering and Technology, 1(2),2012, 76-82.*

[4] Simrandeep Singh Thapar, Paramjeet Singh, Shaveta Rani, “Challenges to the Development of Standard Software Quality Model”, *International Journal of Computer Applications, 49(10),2012,1-7.*

[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, 1(2),2010,100-104.

[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, 3(9), 2011,30-38.*

[7] Aldeida Aleti, Indika Meedeniya, "Component Deployment Optimisation with Bayesian Learning", *ACM Journal, 2011, 11-20*.

[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, 3(4), 2011, 160- 173*.

[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, 5(4),2011, 159-170.*

[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, 43(1), 2010, 853-856.*

[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, 6(1), 2009, 731-740.*

[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, 6(2), 2009,101-106*.

[13] R. Senthil, D. S. Kushwaha, A. K. Misra, "An Extended Component Model and its evaluation for Reliability & Quality", *in Journal of Object Technology, 7(7), 2008, 109-129*.

[14] Yoonjung Choi, Sungwook Lee, Houp Song, Jingoo Park, SunHee Kim, "Practical S/W Component Quality Evaluation Model", *ICACT, 10,2008, 259-264.*

[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, 2008, 158–173*.

[17] Kung-Kiu Lau, Zheng Wang, "Software Component Models", *IEEE Transactions On Software Engineering, 33(10),2007,709-724*.

[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, 2006,444-454*.

[20] Kilsup Lee, Sung Jong Lee, "A Quantitative Software Quality Evaluation Model for the Artifacts of Component Based Development", *Proc. of the 6th IEEE International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005,551-560.*

[21] Simon Haykin, Neural Networks: "A Comprehensive Foundation", *Pearson Education, 2002.*

[22] Xia Cai, Michael R. Lyu, Kam-Fai Wong, Roy KO, "Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes", *IEEE, 2000, 372-379.*

[23] John Grundy, Warwick Mugridge, John Hosking, "Constructing Component-based Software Engineering Environments: Issues and Experiences", *Elsevier Journal of Information and Software Technology, 42(2),2000,117-128.*

[24] Neville I. Churcher, Martin J. Shepperd, "Comments on - A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering, 21(3),1995,263-265*.

[25] R. Geoff Dromey, "A Model for Software Product Quality", *IEEE Transactions on Software Engineering, 21(2), 1995, 146-162*.

[26] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", *Addison Wesley, 1995.*

[27] MatLab Neural Network 2010 Tool Box Product Help.