Performance Enhancing Component Based Software Quality Models

[1]Karnail Singh, [2]Dr. Sushil Garg
[1,2]Dept. of Computer Science, RIMT Institute of Engg. Tech., Mandi Gobingarh, Punjab, India

Abstract
Software component technology is widely seen as the best means of achieving the gains in programmer efficiency, system elasticity, and overall system quality required by the IT revolution. With increasing the software applications and the critical risks due to the low quality software, the importance of high quality software development has been getting more important than ever. Therefore, for quality evaluation it is necessary to remove subjective and traditional evaluation patterns and to accept an objective approach that develops software considering quality characteristics from the beginning stage and performs each step's quality evaluation thoroughly. The model defined and illustrated in this research here provides an explicit process for binding quality-carrying properties into software. These properties in turn imply particular quality attributes. This research work, proposes a quantitative software quality evaluation model with respect to the Component Based Development (CBD) methodology.

Keyword
Component Based Software Engineering, SOM Networks, Design Patterns

I. Introduction

Component-Based Software Engineering (CBSE) provides inherent advantage in software quality, productivity and overall system cost. Component-based software engineering is a process that stressed on the design and construction of computer-based systems using reusable software i.e. "components". CBSE is an approach of software development that depends on reuse [1].
In component-based software engineering process two parallel tracks are running, in which domain engineering occurs concurrently with component-based development. Domain engineering performs the necessary work required to establish a set of software components that can be reused by the software engineer. These components are then

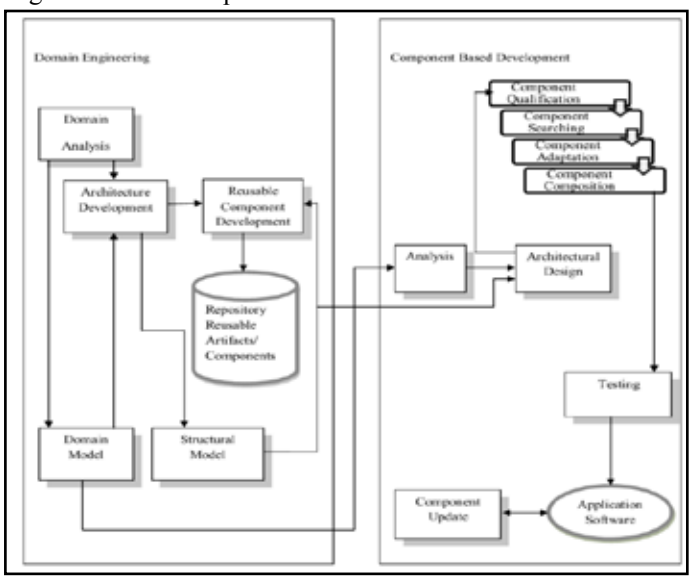

Fig. 1: Component Based Software Engineering Process

carried that separates domain engineering from component-based development.
In addition to COTS components, the CBSE process yields:

1. Qualified Components- These components are assessed by software engineers if they confirm not only functionality, but performance, reliability, usability, and other quality factors of the requirements of the system or product to be built.
2. Adapted Components- That can be adapted to modify (also called mask or wrap) unwanted or undesirable characteristics.
3. Assembled Components- These components can be assembled into any architectural style and can be interconnected with an appropriate infrastructure so that components can be coordinated and managed effectively.
4. Updated Components- These are those components which have power of replacement of existing software as new versions of components become available [2].

Benefits of Module Based Software Engineering

Due to the introduction of high-level programming languages, the advent of component software is one of the most important new developments in the software industry. The advantage of custom software and standard software can be combined in component software. It produces solutions that are more readily maintainable, are better evolvable, can be extended over time, and can be modernized incrementally.

Business Benefits of CBSE

Component based software engineering provides various business recompense, these are

- Short Time to Market- Applications with modules can be delivered synchronously with the business change cycle time.
- Adaptable- Componentized applications are able to respond rapidly to change without forcing costly rewrites of the entire system.
- Supporting Integration- There is often no time or justification to replace legacy applications. New functionality can be incorporated with other packages, presented applications, and data sources.
- Applicable- Componentized applications can align with the business easily. Due to these applications there is no need of doing change in the functionalities of business applications.
- Upgradeable- Improvements to an presented function can be possible without impact to other functions.

Technical Benefits of CBSE

In the view of component users, the technical benefits can be summarized as following:

- Small developers will find that components reduce expenses and lower the barriers to entry in the software market. They can create individual components with the knowledge that they will integrated smoothly with existing software created by larger development shops - they do not have to

reinvent all the functions around them. They can get superior grained amalgamation as a substitute of today's 'band-aid' amalgamation. In addition, they get quicker time to market because the bulk of an application is already there.

- Large Developers, System integrators will use component suites to create enterprise-wide client/server applications in record time. Typically, about 80% of the function they need will be available as off-the-shelf components. The remaining 20% is the value-added they provide. The client/server systems may be less complex to test because of the high reliability of the pre-tested components. The fact that many components are black-boxes reduces the overall complexity of the development process.
- Desktop vendors will use components to assemble applications that target explicit markets. As a substitute of selling monster suites at high prices, they will be able to provide their consumers with what they really want [3].

Cost Associated With CBSE

Component-based software engineering has perceptive appeal. It is assumed that, it should provide a software organization with advantages in quality and timely completion and these parameters should result into cost savings [4]. But these factors are correctly measured when we fully understand what actually can be reused in a software engineering context and after that the costs associated with reuse. It is fundamental that there should be some hard data that can support this. So that it is possible that one should develop a cost/benefit analysis for component reuse.

II. Aspects of CBSE

Some factors are discussed below which are improved by using CBSE

A. Productivity

There is less time spent in creating the schemes, models, documents, code, and data that are required to create a deliverable system, when reusable components are applied throughout the software process. So by doing less efforts in input we can achieve same level of functionality. In this way, productivity is improved [5].

B. Quality

A software component which is developed for reuse is assumed to be right and would contain no defects, in ideal situation. But in reality, formal verification is not carried out regularly, and defects can and do occur. However, defects are consistently found and removed after each reuse and hence component's quality improves as a result. And after reuse, the component becomes defect free.

C. Cost

By calculating the overall cost of the project the net cost savings for reuse is estimated. If Cs, is cost associated with development from scratch, and then subtracting the sum of the costs allied with salvage, Cr, and the actual cost of the software as delivered i.e. C d [6].

Few key points concerning CBSE is mentioned beneath [7]

1. CBSE Process has two parallel sub-process - Domain engineering and CBD.
2. CBD has 4 activities - Qualification, Adaptation, Composition, and Updation.
3. The component based software lifecycle uses V- model for development.
4. The object model generally conforms to one or more standards (OMG/CORBA, COM, DCOM, and Java Bean).
5. Classification schemes help the developer to find and retrieve the reusable components.
6. A component based approach cannot be utilized if the development processes are not adopted according to CBSE principles.
7. CBD has a lot of promises but is not silver bullet.

D. Design Patterns

Design patterns can be categorized basically into 4 types as shown in the figure 2 [8].

1. Creational Patterns

(i). Abstract Factory

Define an abstract class that specifies which objects are to be made. Then implement one concrete class can for each family. Tables or files can also be used to essentially achieve the same thing. Name of the desired classes can be kept in a database and then switches or dynamic class loading can be used for the correct objects.

(ii). Encapsulates

The rules about what our the families objects are that is, which ones go together.

Implementation:

Indicators in Analysis: In Different cases that require different implementations for sets of rules.

Indicators in design: Many polymorphic structures exist that are used in pre-defined combinations. These combinations are defined by there are being particular cases to implement or different needs of client objects

Indication pattern is not being used when it should be: A variable used in several places to determine which the object to instantiate.

Relationships involved: The Abstract Factory object is responsible for coordinating the family objects that the client object needs. The client object has the responsibility for using the objects.

Principle manifested: Encapsulate construction.

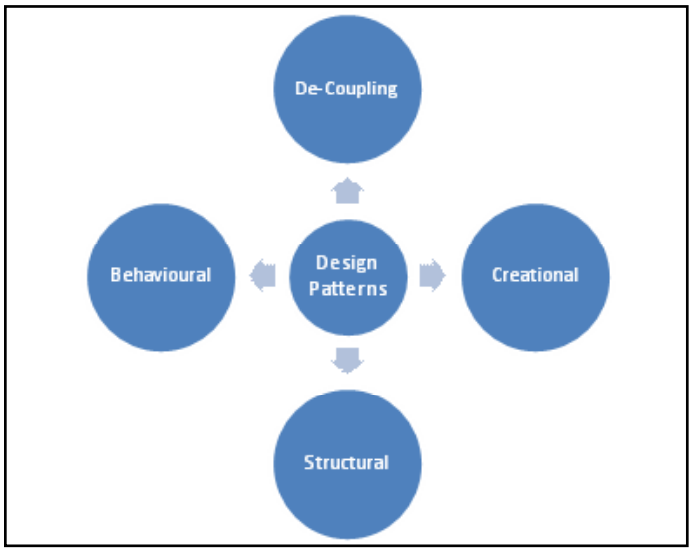

Fig. 2: Types of Design Patterns

- **Builder:** Create a factory object that contains several methods. Each method is called separately and performs a necessary step in the building process. When the client object is through,

it calls a method to get the constructed object returned to it. Derive classes from the builder object to specialize steps.

Encapsulates: Rules for constructing complex objects. Implementation:

Indicators in analysis: Several different kinds of complex objects can be built with the same overall build process, but where there is variation in the individual construction steps.

Indicators in design: You want to hide the implementation of instantiating complex object, or you want to bring together all of the rules for instantiating complex objects.

- **Factory Method:** Have a method in the abstract class that is abstract (pure virtual). The abstract class's code will refer to this method when it needs to instantiate a contained object. Note, however, that it doesn't know which one it needs. That is why all classes derived from this one must implement this method with the appropriate new command to instantiate the proper object.

Encapsulates: How two inheritance hierarchies are related to each other.

Implementation: Indicators in analysis: There are different commonalities whose implementations are coordinated with each other.

Indicators in design: A class needs to instantiate a derivation of another class, but it doesn't know which one. The factory method allows a derived class to make this decision.

Field notes: The Factory method is often used with framework. It is used when the different implementations of one class hierarchy requires a specific implementation to another class hierarchy. Note that a factory method of pattern is not simply method that serves as a factory. The pattern specifically involves the case where the factory is varied polymorphic ally. The Factory Method is very useful when unit testing with the Mock Objects.

Principle manifested: Encapsulate the relationship between class hierarchies.

- **Prototype:** Set up concrete classes of the class needing to be cloned. Each concrete class will be construct itself to the appropriate value (optionally based on input parameters). When a new object is needed, clone an instantiation of this prototypical object.

Encapsulates: The default settings of an object when instantiated.

Indicators in analysis: There are prototypical instances of things.

Indicators in design: When the objects being then instantiated need to look like a copy of a particular object. Allows for dynamically specifying what our instantiated objects look like.

Singleton: Add a static member to the class that refers to the first instantiation of this object (initially it is null). Then, add a static method that instantiates this class if these member is null (and sets this member's value) and then returns the value of this member. Finally, they set the constructor to protected or private so no one can directly instantiate this class and by pass this mechanism.

Encapsulates: That there are only one of these objects allowed.

Indicators in analysis: There are exists only one entity of something in the problem domain that is used by several different things.

Indicators in design: Several different client objects need refer to the same thing and we want to make sure that we don't have more only one of them. You only want to have one of an object but there is no higher object controlling the instantiation of the object in questions.

Field notes: You can get much the same function as Singletons with static method, however they are using the static method is eliminates the possibility of handing future change through polymorphism, and also prevents the object from being passed by reference, serialized, remote, etc. In general, the statics are to be avoided if they are possible. Also, the keep in mind that a state full Singleton is essentially a global, and thus can potentially create coupling to any part of your system from any other part, so they should be used with care. Stateless Singletons do not have this problem.

III. Self Organizing Map (SOM) Neural Network

Self-organizing in network is one of the most fascinating topic in the neural network field. Such network can learn to detect regularities and correlations in the input and adopt their future responses to that input accordingly. The neurons of competitive network learn to recognize groups of similar input vectors [9][10]. Self-organizing maps learn to recognize groups of similar input vectors in such a way that neurons physically near each other in the neuron layer response to similar input vectors. Self-organizing feature map (SOFM) learn to classify input vectors according to how they are grouped in the input space. They differ from competitive layers in neighboring neurons in the self-organizing map learn to recognize neighboring sections of the input space. Thus, self-organizing map learn both the distribution (as do competitive layers) and topology of the input vectors they are trained on. Here a self-organizing feature map network identifies a winning neuron i* using the same procedure as employed by a competitive layer. However, they instead of updating only the winning neuron, all neurons within a certain neighborhood Ni* (d) of the winning neuron are updated using the Coonan rule [11]. Specifically, all such neurons are adjusted as follows:

$$_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$$

or

$$_i\mathbf{w}(q) = (1-\alpha){}_i\mathbf{w}(q-1) + \alpha\mathbf{p}(q)$$

Here the neighborhood Ni* (d) contains the indices for all of the neurons that lie within a radius d of the winning neuron i*.

$$N_i(d) = \{j, d_{ij} \leq d\}$$

Thus, when a vector p is presented, the weights of the winning neuron and its close neighbors move toward p. Consequently, after many presentations, neighboring neurons have learned vectors similar to each other.

The neurons of the layer are arranged in a SOFM physical positions as originally a function of the topology. The grid top function hex top or red top can arrange the neurons in a hexagonal lattice, or random topology. The distances between the neurons are calculated from their positions with a distance function. There are four remote functions, dist, boxiest, lankiest and manliest. Link distance is the most common.

Another version of the SOFM training algorithm called batch, presents all the data on the network before the weights are updated. The algorithm determines a winning neuron for each input vector. Each weight vector moves to the average position of all the input vectors for which there is a winner, or for which it is in the vicinity of a winner.

The concept of neighborhoods is shown in Figure below. The left diagram show two-dimensional neighborhood of radius d = 1 around neuron 13 The diagram below shows an area against radius d = 2.

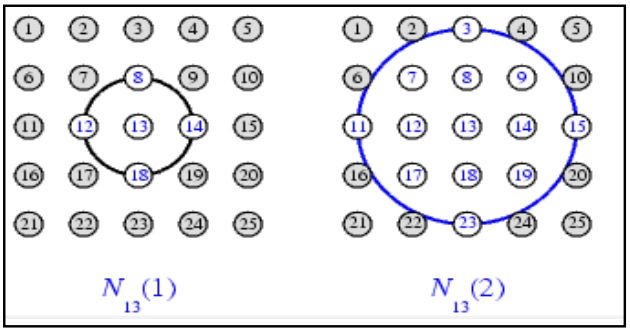

Fig. 3: SOM Training Neighborhoods

These neighborhoods could be written as

$$N_{13}(1) = \{8, 12, 13, 14, 18\}$$

and

$$N_{13}(2) = \{3, 7, 8, 9, 11, 12, 13, 14, 15, 17, 18, 19, 23\}$$

The weights adapt during the learning process based on a competition, i. e. the nearest
(the most similar) neuron of the output layer to the submitted input vector becomes a winner and its weight vector and the weight vectors of its neighboring neurons are adjusted accordingly.
The research work will establish a mechanism through which the content and quality of component models can be measured, which is then applied to find best matrix values for which maximum performance can be achieved. The best values are find out using neural network scheme i.e. Self Organizing Map.
The steps that are followed to obtain the improvement in performance are followed as stated earlier. Number of training data is 21 x 6 i.e. 126 elements. The experiment results for training of neural network in the Mat lab are shown in table 4. In the table 4 values used Training method, No. of training data, No. of epoch taken to converge, No. of output data, time taken to execution of program are shown.

Table 1: The Experimental Results Using Neural Network Analysis

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 seconds
No. of outputs	6

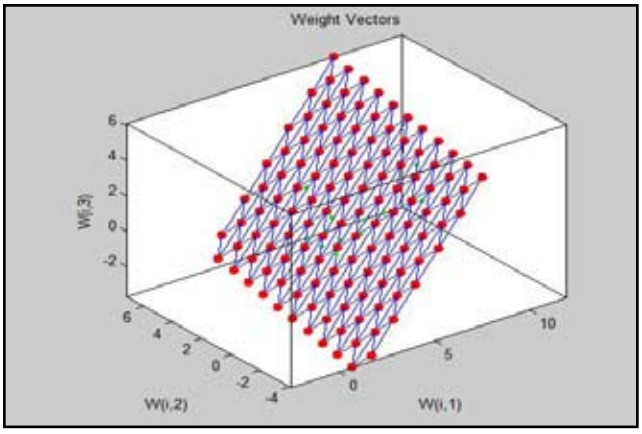

Fig. 4: Initial SOM Vectors for Input Patterns

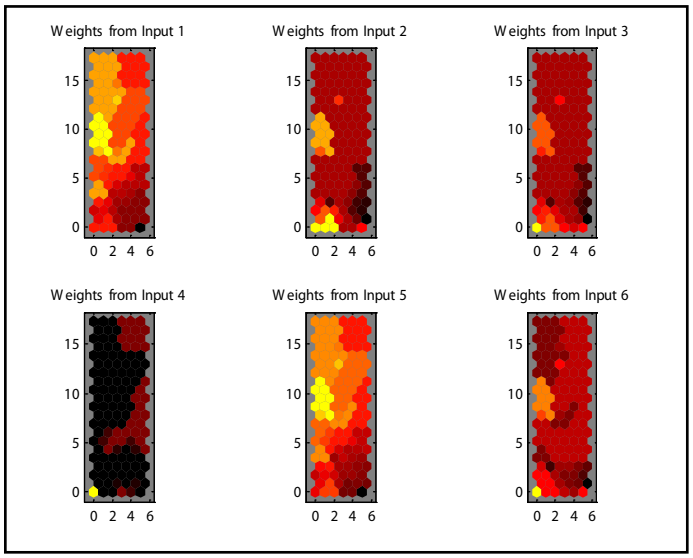

Fig. 5: SOM Weights for All Inputs After Training Phase

IV. Conclusion

The model we have defined and illustrated here provides an explicit process for binding quality-carrying properties into software. These properties in turn imply particular quality attributes. CBSE is a knowledge-intensive activity where collaborators produce and consume knowledge during all the development phase. An effective usage transfer and capture of this knowledge are vital to the survival and the competitiveness of organizations using CBSE. CBSE is adding a lot of value to rapid application development and is actively contributing to better quality software systems.

V. Future Scope

The technique implemented for evaluating and improving performance of Software Component Quality Model is using design patterns. The proposed model can give better values if good amount of data is provided with realistic values. As the model proposed is made for evaluating performance of component based model, historical results of a software company which is using component based development, may provide error free data. If that data is used as an input and results in terms of quality factors like reusability, maintainability, complexity, testability etc. are given as output. By using a supervised neural network may give better results as compared to unsupervised neural network which is used in our research work. We can improve the performance of our system by tuning the inputs from software industry.

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.