ComPARE: A Generic Quality Assessment Environment for Component-Based Software Systems

Xia Cai[1], Michael R. Lyu[1], Kam-Fai Wong[2], Mabel Wong[2]
Dept. of Computer Science and Engineering[1] *Center of Innovation and Technology*[2]
The Chinese University of Hong Kong
{xcai, lyu}@cse.cuhk.edu.hk, kfwong@se.cuhk.edu.hk, mabel@cintec.cuhk.edu.hk

Abstract

Component-based technology is gaining popularity in modern software development. This approach helps reduce development cost and time-to-market, as well as improve maintainability and reliability. One of the key problems in component-based software development is finding a way to certify the quality of individual components and that of the integrated component-based software systems. There are several different techniques which have been developed to describe the predictive relationship between software metrics and the reliability of the software components.

In this paper, we propose a generic quality assessment environment for software components: ComPARE. ComPARE collects various metrics from candidate components including process metrics, static code metrics and dynamic metrics. Also it integrates different models to predict software quality and reliability, and compares the result of different models. With ComPARE, user can select and define their own prediction models and validate these models against the failure data collected in real life. The benchmark models can be established after validation for future use. Finally, prediction results can be visualized and hidden problems can be identified in the source code in the ComPARE environment.

Keyword: Quality assessment tool, component-based software, classification tree model, case-based reasoning, Bayesian Belief Network.

1. Introduction

Component-based software development (CBSD) has become a popular methodology in developing modern software systems. It is generally considered that this approach can reduce development cost and time-to-market, and at the same time are built to improve maintainability and reliability. As CBSD is to build software systems using a combination of components including off-the-shelf components, components developed in-house and components developed contractually, the over quality of the final system greatly depends on the quality of the selected components.

We need to first measure the quality of a component before we can certify it. Software metrics are designed to measure different attributes of a software system and development process, indicating different levels of quality in the final product [1]. Many metrics such as process metrics, static code metrics and dynamic metrics can be used to predict the quality rating of software components at different development phases [1][3]. For example, code complexity metrics, reliability estimates, or metrics for the degree of code coverage achieved have been suggested. Test thoroughness metric is also introduced to predict a component's ability to hide faults during tests [2].

In order to make use of the results of software metrics, several different techniques have been developed to describe the predictive relationship between software metrics and the classification of the software components into fault-prone and non fault-prone categories [4]. These techniques include discriminant analysis [7], classification trees [8], pattern recognition [9], Bayesian network [10], case-based reasoning (CBR) [11], and regression tree models [4]. There are also some prototype or tools [13, 14] that use such techniques to automate the procedure of software quality prediction. However, these tools address only one kind of metrics, e.g., process metrics or static code metrics. Besides, they rely on only one prediction technique for the overall software quality assessment.

In this paper, we propose a Component-based Program Analysis and Reliability Evaluation (ComPARE) to evaluate the quality of software systems in component-based programming technology. ComPARE automates the collection of different metrics, the selection of different prediction models, the formulation of user-defined models, and the validation of the established models according to fault data collected in the development process. Different from other existing tools, ComPARE takes dynamic metrics into account (such as code coverage and performance metrics), integrates them with process metrics and more static code metrics for object-oriented programs (such as complexity metrics, coupling and cohesion metrics, inheritance metrics), and provides different models for integrating these metrics to an overall estimation with higher accuracy.

2. Objective

A number of commercial tools are available for the measurement of software metrics for object-oriented programs. Also there are off-the-shelf tools for testing or debugging software components. However, few tools can measure the static and dynamic metrics of software systems, perform various quality modeling, and validate such models against actual quality data.

ComPARE aims to provide an environment for quality prediction of software components and assess their reliability in the overall system developed using CBSD. The overall architecture of ComPARE is showed in Figure 1. First of all, various metrics are computed for the candidate components, then the users can select and weighing the metrics deemed important to quality assessment. After the models have been constructed and executed (Case Base is used in BBN model), the users can validate the selected models with failure data in real life. If users are not satisfied with the prediction, they can go back to the previous step, re-define the criteria and construct a revised model. Finally, the overall quality prediction can be displayed under the architecture of the candidate system. Results for individual components can also be displayed after all the procedures.

The objective of ComPARE can be summarized as follows:

1. *To predict the overall quality by using process metrics, static code metrics as well as dynamic metrics.* In addition to complexity metrics, we use process metrics, cohesion metrics, inheritance metrics as well as dynamic metrics (such as code coverage and call graph metrics) as the input to the quality prediction models. Thus the prediction is more accurate as it is based on data from every aspect of the candidate software components.
2. *To integrate several quality prediction models into one environment and compare the prediction result of different models.* ComPARE integrates several existing quality models into one environment. In addition to selecting or defining these different models, user can also compare the prediction results of the models on the candidate component

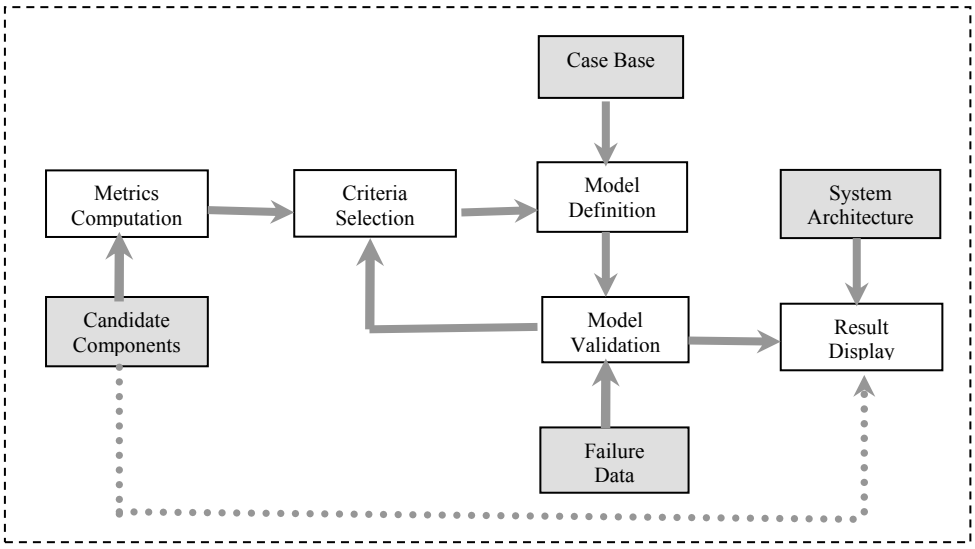

Figure 1. Architecture of ComPARE

and see how good the predictions are if the failure data of the particular component is available.

3. *To define the quality prediction models interactively.* In ComPARE, there are several quality prediction models that users can select to perform their own predictions. Moreover, the users can also define their own model. and validate their own models by the evaluation procedure.
4. *To display quality of components by different categories.* Once the metrics are computed and the models are selected, the overall quality of the component can be displayed according to the category it belongs to. Program modules with problems can also be identified.
5. *To validate reliability models defined by user against real failure data (change report).* Using the validation criteria, the result of the selected quality prediction model can be compared with failure data in real life. The user can redefine their models according to the comparison.
6. *To show the source code with potential problems at line-level granularity. ComPARE* can identify the source code with high risk (i.e., the code that is not covered by test cases) at line-level granularity. This can help the users to locate high risk program modules or portions promptly and conveniently.
7. *To adopt commercial tools in accessing software data related to quality attributes.* We adopt Metamata [5] and Jprobe [6] suites to measure the different metrics for the candidate components. These two tools, including metrics, audits, debugging, as well as code coverage, memory and deadlock detected, are commercially available in the component-based program testing market.

3. Metrics Used in ComPARE

Three different categories of metrics, namely process, static, and dynamic, are computed and collected in CompARE to give an overall quality prediction. We have chosen the most useful metrics, which are widely adopted by previous software quality prediction tools from the software engineering research community. The process metrics we select are listed in Table1 [14].

As we perceive Object-Oriented (OO) techniques are essential in the CBSD approach, we select static code metrics according to the most important features in OO programs: complexity, coupling, inheritance and cohesion. They are listed in Table 2 [5,16]. The dynamic metrics encapsulate measurement of the features of components when they are executed. Table 3 shows the details description of the dynamic metrics.

This set of process, static, and dynamic metrics can be collected from some commercial tools, e.g., Metamata Suite [5] and Jprobe Testing Suite [6]. We will measure and apply these metrics in ComPARE.

4. Models Definition

In order to predict the quality of different software components, several techniques have been developed to classify software components according to their reliability [4]. These techniques include discriminant analysis [7], classification trees [8], pattern recognition [9], Bayesian network [10], case-based reasoning (CBR) [11] and regression tree model [4]. In ComPARE, we integrate five types of models to evaluate the quality of the software components for an overall CBSD system evaluation. User can customize these models and compare the prediction results from different tailor-made models.

4.1 Summation Model

This model gives a prediction by simply adding all the metrics selected and weighted by a user. The user can validate the result by real failure data, and then benchmark the result. Later when new components are included, the user can predict their quality according to their differences from the benchmarks. The concept of summation model can be summarized as the following:

$$Q = \sum_{i=1}^{n} \alpha_i m_i \qquad (1)$$

where m_i is the value of one particular metric, α_i is its corresponding weighting factor, n is the number of metrics, and Q is the overall quality mark.

Metric	Description
Time	Time spent from the design to the delivery (months)
Effort	The total human resources used (man*month)
Change Report	Number of faults found in the development

Table 1. Process Metrics

Abbreviation	Description
Lines of Code (LOC)	Number of lines in the components including the statements, the blank lines of code, the lines of commentary, and the lines consisting only of syntax such as block delimiters.
Cyclomatic Complexity (CC)	A measure of the control flow complexity of a method or constructor. It counts the number of branches in the body of the method, defined by the number of WHILE statements, IF statements, FOR statements, and CASE statements.
Number of Attributes (NA)	Number of fields declared in the class or interface.
Number Of Classes (NOC)	Number of classes or interfaces that are declared. This is usually 1, but nested class declarations will increase this number.
Depth of Inheritance Tree (DIT)	Length of inheritance path between the current class and the base class.
Depth of Interface Extension Tree (DIET)	The path between the current interface and the base interface.
Data Abstraction Coupling (DAC)	Number of reference types that are used in the field declarations of the class or interface.
Fan Out (FANOUT)	Number of reference types that are used in field declarations, formal parameters, return types, throws declarations, and local variables.
Coupling between Objects (CO)	Number of reference types that are used in field declarations, formal parameters, return types, throws declarations, local variables and also types from which field and method selections are made.
Method Calls Input/Output (MCI/MCO)	Number of calls to/from a method. It helps to analyze the coupling between methods.
Lack of Cohesion Of Methods (LCOM)	For each pair of methods in the class, the set of fields each of them accesses is determined. If they have disjoint sets of field accesses then increase the count P by one. If they share at least one field access then increase Q by one. After considering each pair of methods, LCOM = (P > Q) ? (P - Q) : 0

Table 2. Static Code Metrics

Metric	Description
Test Case Coverage	The coverage of the source code when executing the given test cases. It may help to design effective test cases.
Call Graph metrics	The relationships between the methods, including method time (the amount of time the method spent in execution), method object count (the number of objects created during the method execution) and number of calls (how many times each method is called in you application).
Heap metrics	Number of live instances of a particular class/package, and the memory used by each live instance.

Table 3. Dynamic Metrics

4.2 Product Model

Similar to the summation model, the product model multiplies all the metrics selected and weighted by the user and the resulting value indicates the level of quality of a given component. Similarly, the user can validate the result by real failure data, and then determine the benchmark for later usage. The concept of product model is shown as the following:

$$Q = \prod_{i=1}^{n} mi \qquad (2)$$

where m_i is the value of one particular metric, n is the number of metrics, and Q is the overall quality mark. Note that m_is are normalized as a value which is close to 1, so that none of them will dominate the result.

4.3 Classification Tree Model

Classification tree model [8] is to classify the candidate components into different quality categories by constructing a tree structure. All the candidate components are leaves in the tree. Each node of the tree represents a metric (or a composed metric calculated by other metrics) of a certain value. All the children of the left sub tree of the node represent those components whose value of the same metric is smaller than the value of the node, while all the children of the right sub tree of the node are those components whose value of the same metric is equal to or larger than the value of the node.

In ComPARE, a user can define the metrics and their value at each node from the root to the leaves. Once the tree is constructed, a candidate component can be directly classified by following the threshold of each node in the tree until it reaches a leaf node. Again, the user can validate and evaluate the final tree model after its definition. Below is an example of the outcome of a tree model. At each node of the tree there are metrics and values, and the leaves represent the components with certain number of predicted faults in the classification result.

4.4 Case-Based Reasoning Model

Case-based reasoning (CBR) has been proposed for predicting the quality of software components [11]. A CBR classifier uses previous "similar" cases as the basis for the prediction. Previous cases are stored in a case base. Similarity is defined in terms of a set of metrics. The major conjecture behind this model is that the candidate component that has a similar structure to the components in the case base will inherit a similar quality level.

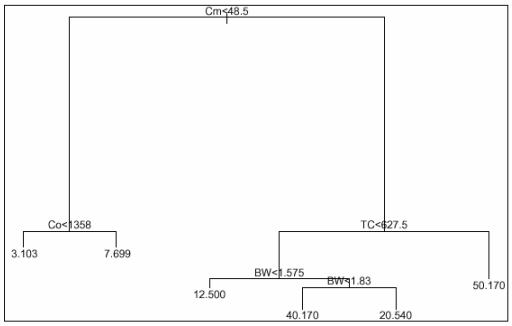

Figure 2. An example of classification tree model

A CBR classifier can be instantiated in different ways by varying its parameters. But according to the previous research, there is no significant difference in prediction validity when using any combination of parameters in CBR. So we adopt the simplest CBR classifier modeling with Euclidean distance, z-score standardization [11], but no weighting scheme. Finally, we select the single, nearest neighbor for the prediction purpose.

4.5 Bayesian Network Model

Bayesian networks (also known as Bayesian Belief Networks, BBN) is a graphical network that represents probabilistic relationships among variables [10]. BBNs enable reasoning under uncertainty. Besides, the framework of Bayesian networks offers a compact, intuitive, and efficient graphical representation of dependence relations between entities of a problem domain. The graphical structure reflects properties of the problem domain directly, which provides a tangible visual representation as well as a sound mathematical basis in Bayesian probability [12]. The foundation of Bayesian networks is the following theorem known as Bayes' Theorem:

$$P(H|E,c) = \frac{P(H|c)P(E|H,c)}{P(E|c)} \qquad (3)$$

where H, E, c are independent events, P is the probability of such event under certain circumstances.

With BBNs, it is possible to integrate expert beliefs about the dependencies between different variables and to propagate consistently the impact of evidence on the probabilities of uncertain outcomes, such as "unknown

component quality". Details of the BBN model for quality prediction can be found in [10]. Users can also define their own BBN models in ComPARE and compare the results with other models.

5. Operations in ComPARE

As a generic quality assessment environment for component-based software system, ComPARE suggests eight major functional areas: File Operations, Selecting Metrics, Selecting Criteria, Model Selection and Definition, Model Validation, Display Result, Windows Switch, and Help System. The details of some key functions are described in the following sections.

5.1 Selecting Metrics

User can select the metrics they want to collect for the opened component-based system. There are three categories of metrics available: process metrics, static metrics and dynamic metrics. The details of these metrics have shown in previous section.

5.2 Selecting and Weighing Criteria

After computing the different metrics, users need to select and weigh the criteria on these metrics before using them in the reliability modeling. Each metric can be selected or omitted, and if selected, be marked with the weight between 0 and 100%. Such information will be used as input parameter later in the quality prediction models.

5.3 Models Selection and Definition

The Models operations allow users to select or define the model they would like to perform in the evaluation. The users should give the probability of each item related to the overall quality of the candidate component.

5.4 Model Validation

Model validation allows comparisons between different models and with respect to actual software failure data. It facilitates the users to compare the different results based on chosen subset of the software failure data under certain validation criteria. The comparisons between different models in their predictive capability are summarized in a summary table. Model Validation operations are activated only when the software failure data are available.

6. Prototype

Under the framework that we have described, we prototyped a specific version of ComPARE which targets software components developed by the Java language. Java is one of the most popular languages used in off-the-shelf components development today, and it is a common language binding in the three standard architecture of component-based software development: CORBA, DCOM and Java/RMI.

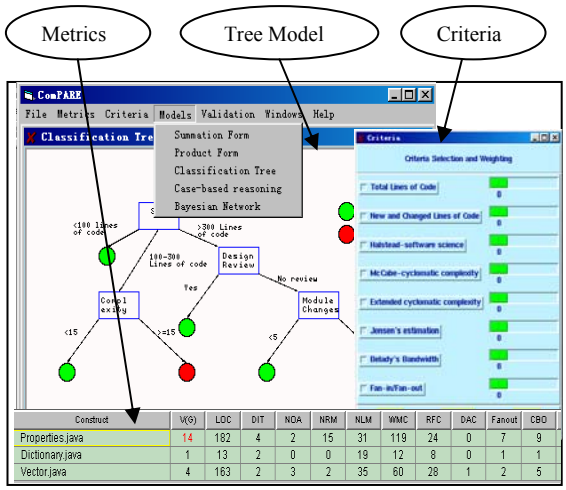

Figure 3. GUI of ComPARE for metrics, criteria and tree model

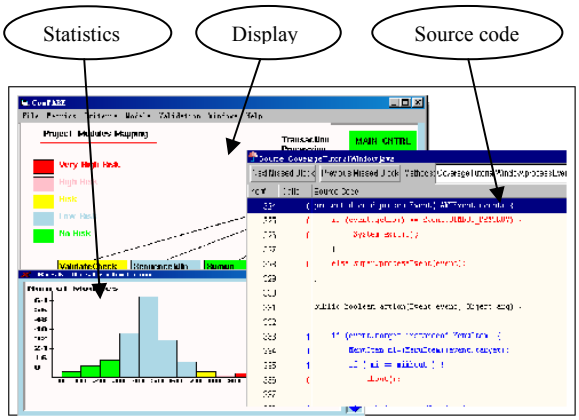

Figure 4. GUI of ComPARE for prediction display, risky source code and result statistics

Figure 3 and Figure 4 show screen dumps for the described ComPARE prototype tool. It can be seen that the computation of various metrics for software components and application of quality prediction models is a straightforward process. Users also have flexible choices in

selecting and defining different models. The combination of simple operations and a variety of quality models makes it easy for users to identify an appropriate prediction model for a given CBSD system with its included components.

7. Conclusions

In this paper, we propose a generic quality assessment environment for software components: ComPARE. ComPARE is new in that it collects metrics of more aspects, including process metrics, static code metrics, and dynamic metrics for software components, integrates reliability assessment models from different techniques used in current quality prediction area, and validates these models against the failure data collected in real life. ComPARE can be used to assess real-life off-the-shelf components and to evaluate and validate the models selected for their evaluation. The overall CBSD system can then be composed and analyzed seamlessly.

In summary, ComPARE can be an effective environment to promote component-based program construction with higher reliability evaluation and proper quality assurance.

Acknowledgement

The work described in this paper was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region (Project No. CUHK4193/00E).

References

[1] M.R.Lyu (ed.), *Handbook of Software Reliability Engineering,* McGraw-Hill, New York, 1996.

[2] J.Voas and J.Payne, "Dependability Certification of Software Components," *The Journal of Systems and Software,* 52, pp.165-172, 2000,

[3] N. E. Fenton and N. Ohlsson, "Quantitative Analysis of Faults and Failures in a Complex Software System," *IEEE Transactions on Software Engineering,* SE-26(8), pp.797–814, Aug. 2000.

[4] S.S.Gokhale and M.R.Lyu, "Regression Tree Modeling for the Prediction of Software Quality," *Proceedings of the Third ISSAT International Conference on Reliability and Quality in Design,* Anaheim, California, March 1997.

[5] http://www.metamata.com, Jan. 2001.

[6] http://www.klgroup.com, Jan. 2001.

[7] J.Munson and T.Khoshgoftaar, "The Detection of Fault-Prone Programs," *IEEE Transactions on Software Engineering,* SE-18(5), May 1992.

[8] A. A. Porter and R. W. Selby, "Empirically Guided Software Development Using Metric-Based Classification Trees," *IEEE Software,* pp. 46-53, Mar.1990.

[9] L.C.Briand, V.R.Basili and C.Hetmanski, "Developing Interpretable Models for Optimized Set Reduction for Identifying High-Risk Software Components," *IEEE Transactions on Software Engineering,* SE-19(11), pp.1028-1034, Nov.1993.

[10] N.E.Fenton and M.Neil, "A Critique of Software Defect Prediction Models," *IEEE Transactions on Software Engineering,* SE-25(5), pp.675-689, Oct. 1999.

[11] K.E.Emam, S.Benlarbi, N.Goel and S.N.Rai, "Comparing Case-Based Reasoning Classifiers for Predicting High Risk Software Components," *The Journal of systems and Software,* 55, pp.301-320, 2001.

[12] http://www.hugin.com, Jan. 2001.

[13] M.R.Lyu, J.S.Yu, E.Keramidas and S.R.Dalal, "ARMOR: Analyzer for Reducing Module Operational Risk," *Proceedings of Twenty-Fifth International Symposium on Fault-Tolerant Computing* (FTCS-25), pp.137-142, 1995.

[14] A.A.Keshlaf and K.Hashim, "A Model and Prototype Tool to Manage Software Risks," *Proceedings of the First Asia-Pacific Conference on Quality Software,* pp.297-305, 2000.

[15] J.F.Patenaude, E.Merlo, M.Dagenais and B.Lague, "Extending Software Quality Assessment Techniques to Java Systems," *Proceedings of the Seventh International Workshop on Program Comprehension,* pp.49-56, 1999.

[16] T.Systa, Y.Ping and H.Muller, "Analyzing Java Software by Combining Metrics and Program Visualization," *Proceedings of the Fourth European Software Maintenance and Reengineering,* pp.199 –208, 2000.