Vol.17 No.6 J. Comput. Sci. & Technol. Nov. 2002

# A Model-Based Approach to Object-Oriented Software Metrics

MEI Hong (梅 宏), XIE Tao (谢 涛) and YANG Fuqing (杨芙清)

*Department of Computer Science and Technology, Peking University, Beijing 100871, P.R. China*

E-mail: {meih, xietao, yang}@cs.pku.edu.cn

Received October 12, 2000; revised March 11, 2002.

**Abstract** The need to improve software productivity and software quality has put forward the research on software metrics technology and the development of software metrics tool to support related activities. To support object-oriented software metrics practice effectively, a model-based approach to object-oriented software metrics is proposed in this paper. This approach guides the metrics users to adopt the quality metrics model to measure the object-oriented software products. The development of the model can be achieved by using a top-down approach. This approach explicitly proposes the conception of absolute normalization computation and relative normalization computation for a metrics model. Moreover, a generic software metrics tool — Jade Bird Object-Oriented Metrics Tool (JBOOMT) is designed to implement this approach. The parser-based approach adopted by the tool makes the information of the source program accurate and complete for measurement. It supports various customizable hierarchical metrics models and provides a flexible user interface for users to manipulate the models. It also supports absolute and relative normalization mechanisms in different situations.

**Keywords** software metrics, object-oriented metrics, software quality model, software reuse

## 1 Introduction

Focus on software quality control has spurred the object-oriented approach to software development and increased the demand for software metrics. Object-oriented metrics is an integral part of object technology and plays an important role in software development. With the maturation of object-oriented concepts and methods, object-oriented programming languages have been used more and more widely. However, the theories, techniques and tools of object-oriented software metrics are still under developing. The research on object-oriented software metrics has been one of the focuses on software metrics. Generally, object-oriented software metrics needs to solve the following problems: What is the procedure of the measurement of the object-oriented software products? How to design and implement a metrics tool to support the activities of the object-oriented metrics effectively? How to apply the object-oriented metrics technology in the software development process, especially in the component-based software development process?

So far, these problems have not been solved satisfactorily. The attributes of software can be divided into internal and external ones. Internal attributes typically describe structural complexity, such as the size of software, complexity of control flow, inter-component coupling, etc. Generally, they are clearly defined and can be measured objectively. External attributes typically represent those exterior factors related with people and environment, such as complexity, maintainability, readability, etc. Generally, only the metrics of external attributes can really provide the reliable information required by people, which predict the behaviors of software development. However, the external attributes often lack explicit definition, and cannot be measured directly and objectively. Therefore, it is a feasible way to obtain the values of external attributes by indirectly measuring the internal attributes.

This effort gets support from the National '863' High-Tech Program and National Natural Science Foundation of China under the Grant Nos.2001AA110236 and 60043002 respectively. It also gets support from National Excellent Young Scientists Foundation (Grant No.60125206) and Research Fund for the Doctoral Program of Higher Education of MOE (Grant No.20010001001).

One of the tasks of metrics research is to formulate the rational relation between external and internal attributes, which is also called the metrics model or quality model.

As early as in 1977 and 1978, McCall quality model[1] and Boehm quality model[2] were proposed respectively. In recent years, ISO 9126 model has been proposed[3]. It improves the McCall model and defines six factors but does not elaborate on criteria and metrics layers. REBOOT model[4] is composed of one quality model and one reusability model, which give detailed definition and description for each layer. However, although the research on software quality models has been carried out for a long time, there has not been a common quality model yet which can be accepted widely. There are several reasons. The dominant one is that the external attributes of software lack explicit and detailed definition and the various existing definitions for them are not united. In addition, the associations between internal and external attributes are hard to validate theoretically, for there exists no common verification principle to judge whether the models are reasonable or not. Many software metrics researchers validate and analyze the proposed metrics models only according to some verification principles they are concerned about.

According to various metrics domains and metrics users, the attributes and their association involved in the metrics models should be adapted accordingly. By validating empirically the metrics models in practice, metrics users can adjust and improve the models, making them fulfil the requirements in specific domains and for specific users. Therefore, an effective approach suitable for this measurement process is greatly wanted to facilitate users to perform the metrics activities. We propose a model-based approach to object-oriented software metrics. This approach clearly defines the activities and procedure of the measurement process. It explicitly proposes the conception of absolute normalization computation and relative normalization computation for metrics model, which can be applied to different situations respectively.

Moreover, we have developed a generic software metrics tool — Jade Bird Object-Oriented Metrics Tool (JBOOMT) to implement this approach. This tool adopts parser-based approach to analyze the source code, thus providing an accurate and complete source for metrics collection. Due to its generic characteristic, the tool provides a flexible user interface for metrics users to filter and tune in calculating and reviewing the metrics data. Users can customize the metrics model based on the specific requirements of a measurement task in hand. Therefore the tool can effectively support customizing, browsing and comparing the hierarchical metrics models.

This paper is organized as follows. Section 2 introduces the model-based approach to object-oriented software metrics. Section 3 presents an overview of JBOOMT and the generic characteristics of each layer in JBOOMT architecture. Section 4 discusses the application of this approach to Jade Bird Component Library and Domain Engineering and compares these two ways of applications. Section 5 presents an example to demonstrate this approach. Finally, Section 6 concludes the paper.

## 2 The Proposed Approach to Object-Oriented Software Metrics

### 2.1 Steps in the Approach

In order to assist people to measure the quality of object-oriented software products effectively, a model-based approach is proposed as illustrated in Fig.1.

There are five steps in this approach:

(1) Selecting the measured entities. The measured entities are classified into system, class, and method according to their scopes.

(2) Constructing the model. According to the top-bottom development process, the metric models are constructed for selected entities.

(3) Computing the model. In order to make different metrics methods fit together to derive the values of the associated high-level quality criteria, it is necessary to normalize the metrics values. The normalization can be classified into absolute and relative normalization.

(4) Validating and comparing the model. By comparing the indirect metrics values of the model and direct metrics values of the high-level quality attributes in the model, users can validate the effectiveness of the model. Moreover, users can compare the values of different models to choose and improve the models.

(5) Improving and using the model. The models are validated and improved iteratively in practice. They are continually improved by utilizing the feedback information in use.

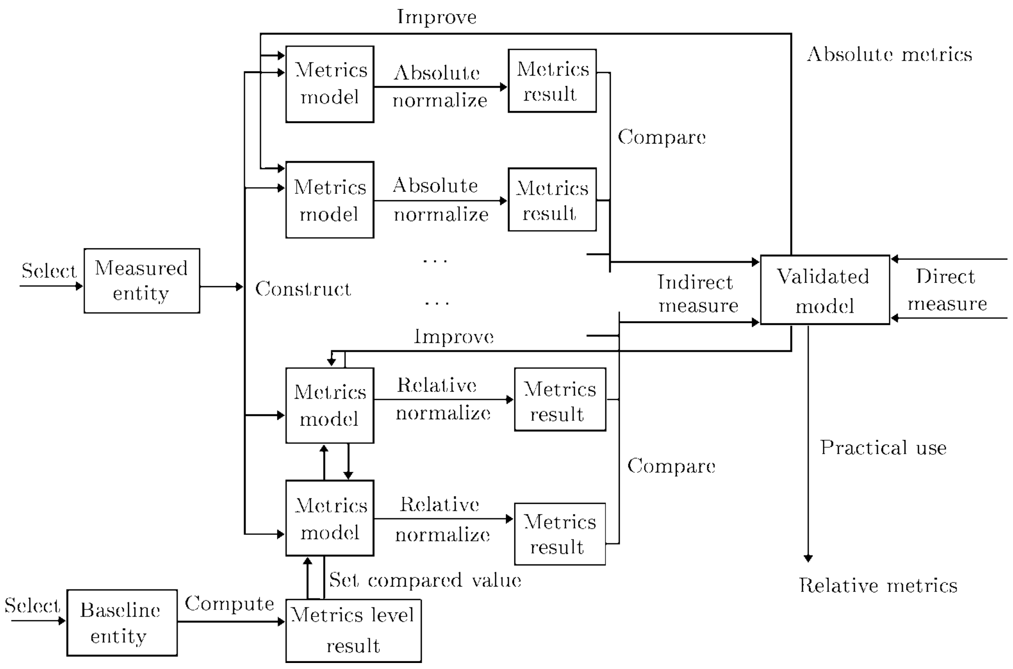

Fig.1. Summary illustration of the measurement process.

## 2.2 Selecting the Measured Entities

Currently most quality metrics models do not explicitly specify the scope level of the measured entity, failing to differentiate the entities of method, class or project level. They mix the metrics of different level entities in the same metrics model, which makes the implementation and computation of the model difficult.

It is necessary to distinguish different metrics specified for different entity levels and construct different level models for these metrics. Indeed it is possible that some metrics values of the higher level entities can be derived by averaging those for lower level entities which comprise them. For example, the method complexity metrics of certain class can be obtained by averaging the complexity metrics of those methods in this class. Similarly, the method complexity metrics of certain system equals the average value of the method complexity metrics of those classes in this system.

However, it is still required to distinguish the metrics models at different entity levels, because some metrics types are only meaningful for certain level entity and the higher level entities generally possess the metrics types which have no corresponding ones at the lower levels. For instance, the hierarchy depth metrics at the class level do not exist at the method level. Therefore, during the first phase of the measurement process, the measured entities have to be specified. Our approach mainly focuses on the measurement of entities of object-oriented software products, which are classified as method, class and system.

## 2.3 Constructing the Model

Generally, the most popular quality metrics models are hierarchical. In this type of models, the values of the higher level nodes can be acquired by computing those values of the lower level nodes which are associated with them. Only the nodes of metrics type at the bottom of the models are directly computed from the information in the source code.

We propose the following steps for top-down model development:

(1) Specifying the normalization type of the model: relative or absolute normalization.

(2) Specifying the integral software quality attributes concerned by specific metrics users as the top layer of the model, such as quality, reusability, etc.

(3) Including those quality factors which influence the quality attributes into the model as the second layer.

(4) Assigning the weight values to the association between the quality attributes and quality factors according to the extent that the quality factors are concerned and emphasized by metrics users. Generally these weight values are positive.

(5) Specifying those quality criteria that influence the quality factors and reflect the internal structural attributes.

(6) Assigning the influence weight values to the association between quality factors and quality criteria based on experience, rationales and empirical study. If the model is an absolute normalization type, the weight values are positive. If it is a relative normalization type, the weight values assigned to them are positive when the influences of quality criteria on quality factors are positive. An example is that the weight value between modularity and adaptability is positive. When the influences are negative, the weight values are also negative. For example, the weight value between structure complexity and understandability is negative.

(7) Specifying those metrics methods which have impacts on the quality criteria.

(8) Assigning the influence weight values to the association between quality criteria and metrics methods according to experience, rationales and empirical study. These weight values are generally positive.

As shown in the above steps, the quality attributes, factors and their associations at the higher levels of the model are greatly influenced by the subjective inclination of the metrics users and are therefore easy to change. However, the bottom level associated with the specific quality factors are specified based on some experience, rationales and empirical study acknowledged by software academia and industry. They are relatively fixed.

## 2.4 Computing the Model

Unlike the model development, the computation of the model is bottom-up. First the software products are analyzed and the program information is attained. Then according to the association between layers and the weight values assigned to them, the values of the entities at the higher level of the model can be computed bottom-up layer by layer.

During the weighted computation process, the normalization problem should be solved. It is unreasonable or meaningless to add the values of different types of metrics together directly because they have different metrics value domains. For example, the line of code for a class might be 50 and the number of its subclass might be 5. Before these two metrics values are normalized into the values with the common reference domain, they cannot be combined or added together.

There are two ways of normalizing the measured values. One is absolute normalization by which the measured values can be normalized according to some predefined empirical reference values. REBOOT metrics models adopt this type of normalization[4]. The value of each item in the model should be normalized so that it will yield a value within certain scope, such as between 0 to 1. A value close to 0 indicates that the measured characteristic may cause problems, while a value close to 1 indicates that the corresponding characteristic is kept inside its limits. Absolute normalization should be performed before adding the metrics values at the bottom level in the model to calculate items at the higher level.

To implement this normalization, a set of threshold values or parameter values should be defined in advance so that the measured values can be normalized to yield values ranging from 0 to 1 according to the following three specified types of metrics[4].

1) Upper limit metrics which is characterized by a break-off value $a$ and a 50% limit $b$. When the measured value $m$ reach the break-off value $a$, the normalized value $M$ will start to decrease. For instance, the inheritance depth of class and McCabe's cyclomatic number fall into this type. From the qualitative perspective, the lower the measured value is, the better the quality of the evaluated object is and it means that the normalized value is much closer to value 1. Fig.2 shows the relation between the computed metrics $M$ and the measured software measurement $m$.

The formula for this type of metrics is described as

$$M(m) = \frac{1}{1 + e^{x\times(m-y)}}$$

In this formula, the parameters $x$ and $y$ can be calculated from these two relations: $M(a) = 0.99$ and $M(a + b) = 0.5$. In this case, it is necessary to demarcate the thresholds of $a$ and $b$ according to the specific metrics.

2) Optimum value metrics which is characterized by center value $a$ and 50% limits at $a + b$ and $a - b$. While concerning quality, when the measured value is too small or too large, the quality of the object under discussion is not good enough. A measured value around $a$ is viewed as an ideal value. Some examples in this case are the average member function size measured by LOC per member function and relative number of comments in a method. Fig.3 illustrates the relation between the computed metrics $M$ and the measured software measurement $m$.

The formula for this second type of metric is defined as

$$M(m) = e^{-x\times(m-a)^z}$$

The parameter $z$ specifies the squareness of the graph. And parameter $x$ also can be calculated from these two equations: $M(a + b) = 0.5$ and $M(a - b) = 0.5$. Similarly, the thresholds of $a$ and $b$ should also be demarcated according to the specific metrics.

3) Linear dependency metrics which is characterized by the intercept $b$ of the measured value $m$ coordinate. It is a simpler type of metrics whose typical examples are the inheritance generality and the relative number of system-dependent code lines. The formula for this type of metrics is simply defined as $M(m) = 1 - m/b$. Fig.4 illustrates the relation between the computed metrics $M$ and the measured software measurement $m$.

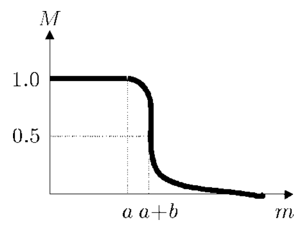

Fig.2. Upper limit metrics.

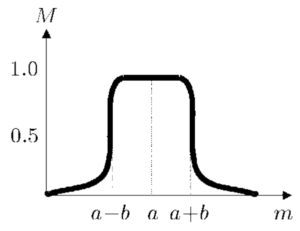

Fig.3. Optimum value metrics.

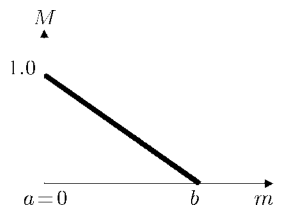

Fig.4. Linear dependency metrics.

The quantitative and qualitative evaluation process of the specific metrics value is performed during the absolute normalization. When calculating the values of the higher level items, the weight values assigned to association only reflect the extent of the influence of the lower level items on higher level items without caring whether these influences are positive or negative. Therefore, in the absolute normalization type of model, all the weight values are positive.

According to the above three types of formula and parameters, the metrics values in the metrics layer can be normalized to a normalized value between 0 and 1. And then the values of the higher level items in the model can be derived from the weighted sum of the values of those items associated with it. However, in practice, it is hard to define a set of fixed threshold parameters for each type of metrics. For instance, it is hard to decide how many classes in a project would be best, for these threshold parameters are affected by the domain of the measured entities and the subjective experience of the metrics users. Moreover, it is possible that certain metrics associate with several quality criteria and the influences of this metrics on them may be inconsistent, even reverse. It is not reasonable to decide the influences of metrics values on the associated quality criteria before investigating the criteria. For example, the increase of the design size may decrease the understandability, but increase the reusability. Therefore it is necessary to decide the influences of the specific metrics on those associated quality criteria respectively, not as a whole.

In addition, usually the value of the metrics is not an absolute but a relative quantity. Since there exists no common metrics unit in software metrics field, for most types of metrics, it is hard to find an absolute reference point as the common fixed reference point for the measured entities. It is hard to specify the zero scale as the baseline to measure the software like a physical gauge. The absolute normalization of certain metrics is to consider certain aspects of the measured entity's attributes, referring to the absolute reference point, the values in the threshold parameter sets, and then normalize the metrics values to the evaluation extent values of good or bad. These extent values are combined and yield values of the higher level external attributes.

The other normalization type is relative normalization. QMOOD model adopts the relative normalization[5]. While performing the relative normalization, generally a group of measured entities are evaluated, and one of them is set as the compared baseline entity. The metrics values in the bottom layer of the model for this baseline entity are compared by the corresponding values of other entities in this group. Relative normalization is to divide the metrics values in the bottom layer of the model by the corresponding values of the baseline entity. In fact, the relative normalized value is a ratio value. After each metrics value has been normalized to a ratio value, the values of items at the higher level of the model can be calculated by weighted sum of the values of the lower level associated items. The influences of metrics on quality criteria and of the quality criteria on quality factors are reflected in the weight values assigned to the associations. If the lower level item has positive influences on the higher level items, the weight value of the association is positive. If the influences are negative, the weight value is negative. The absolute value of the weight value indicates the extent of the influence. Therefore if the normalized values and the values of the higher level items in the model are relatively higher, then its evaluation is relatively better. The lower, the worse.

Relative normalization also has shortcomings. First, if the metrics value of the baseline entity is zero, then the approximation process has to be done when performing the relative normalization. However, it is also very difficult to assign an approximate value to it, for the scale of different metrics types are different. Moreover, during the normalization, it assumes that the relation between the higher level quality criteria and metrics values is linear dependent. Judging from experience, it assumes that if the values of quality criteria increase when the metrics values increase and decrease when the metrics values decrease, the weight value assigned to their association is positive. Otherwise, the weight value is negative. Although it avoids the difficulty of defining the threshold parameters, it does not take into account the fact that some metrics types are not linear dependency metrics, but upper limit metrics or optimum value metrics.

Summarily, both absolute and relative normalizations have their advantages and disadvantages respectively. Each has its appropriate situation to apply. When making comparison among several measured entities, only those entities with similar functional requirements can be compared meaningfully[5]. It is very difficult to compare the quality and reusability of those components with different functionality or in different domains. Therefore, in practice, metrics users usually compare the quality of the entities with similar functionality and set one of the entities as the baseline entity. However, in some other situations, the measured entities may still implement different functionality and these entities are relatively independent and not comparable, or there is only one measured entity. In these cases, it would be better to adopt the absolute normalization to compute the models.

### 2.5 Validating and Comparing the Model

While put into practice, metrics model can be validated empirically to provide the basis for the improvement of the model. Direct metrics is to measure the attributes of the software products without the necessity to measure other attributes. Indirect metrics is to measure the attributes of the software products by measuring one or more other attributes. Software products possess external and internal attributes and a metrics model exhibits a form of reasonable association between external and internal attributes.

The computation of the model is to measure the value of the external attributes (the top quality attributes layer) indirectly by measuring the internal attributes (the bottom metrics layer). Generally speaking, this measurement is objective and does not alter with different metrics users. At the same time, in order to validate the model, metrics users can directly measure the quality attributes at the top level of the model by grading the quality attributes after analyzing the software products. This is the subjective measurement. In order to reduce the subjective error, the software product can be evaluated by several metrics users and these subjective metrics values can be averaged to yield the average direct metrics value. Finally, the efficiency of the model can be validated by comparing the difference between the direct metrics value and indirect metrics value.

As the metrics model develops, the model developer needs to select several candidate models for comparison and finally decide on the adopted model according to the result of validation and comparison. Moreover, when adjusting the metrics model, it is also necessary to compare the result values of the models before and after the adjustment to validate the effect of the adaptation.

### 2.6 Improving and Using the Model

Before the models are put into practice widely, they need to be validated and improved repeatedly till they meet the requirements of the users satisfactorily. The validated models can be used by software engineers, project managers, researchers and other interested people to evaluate the quality of object-oriented software products. While these models are being used, some feedback information and data can be collected and used as the basis for further improvement.

## 3 JBOOMT — A Metrics Tool Supporting the Approach

Jade Bird Object-Oriented Metrics Tool (JBOOMT) is designed and developed to implement the above model-based approach to assess the quality of object-oriented software. The tool makes it easy for the hierarchical layered metrics models to be constructed, calculated and presented.

As shown in Fig.5, the metrics measurement consists of four phases: constructing, analyzing, calculating and displaying. Correspondingly, the tool is divided into four parts.

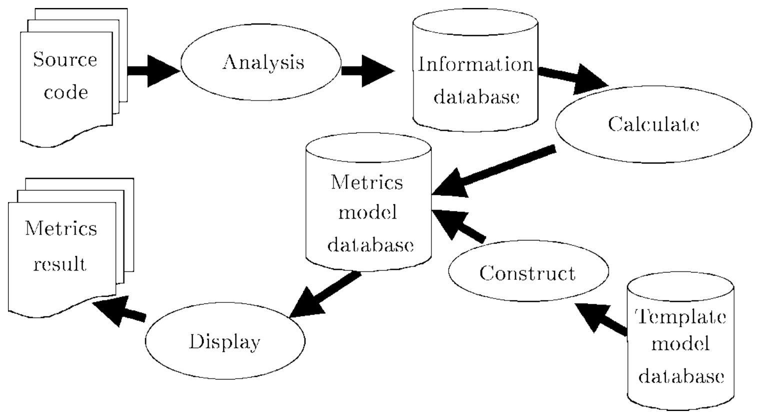

Fig.5. The workflow of JBOOMT.

In the constructing phase, metrics users customize the metrics model based on the implemented metrics or adapt the template model stored in the template model database.

In the analyzing phase, the analysis front end analyzes source code, extracts program information and stores it in the program information database through the database server[6]. A front end which includes both the interface and implementation part of the program is developed to analyze the source code syntactically and semantically.

In the calculating phase, after users select some models in the model database, all the values of the model are calculated from the information database and then are stored in the metrics model database. In the tool, the model database is used to store the definition of the hierarchical models and the result values of the calculated hierarchical model.

In the displaying phase, the display part loads the metrics data from the metrics model database and provides visual presentation such as chart, graph or illustration to display the metrics results.

Major parts and layered architecture of JBOOMT is illustrated in Fig.6. JBOOMT is composed of four layers: the analysis layer, the construction layer, the calculation layer and the representation layer. Each layer has some attractive characteristics so that users can tailor the metrics model in practice. The generic characteristics are especially helpful. The tool provides a finer user control and a flexible way to manipulate. Metrics users can customize the metrics model in the tool actively as they wish, rather than passively use the fixed models built in the tool. Each layer and its characteristics are explained as follows.

• Analysis Layer C++ Parser

JBOOMT adopts the parser-based approach to analyze C++ program source code based on the compiler technology. As is shown in the top part of Fig.6, the front end of tool analyzes C++ programs statically in the way of incremental parsing, extracts program information according to a conceptual model formed with Enhanced Entity Relationship model, and stores the information through the database server in a relational database — program information database. The front end only needs to know the exact syntax of the C++ language, which means that when the target language changes, it only affects the front end.

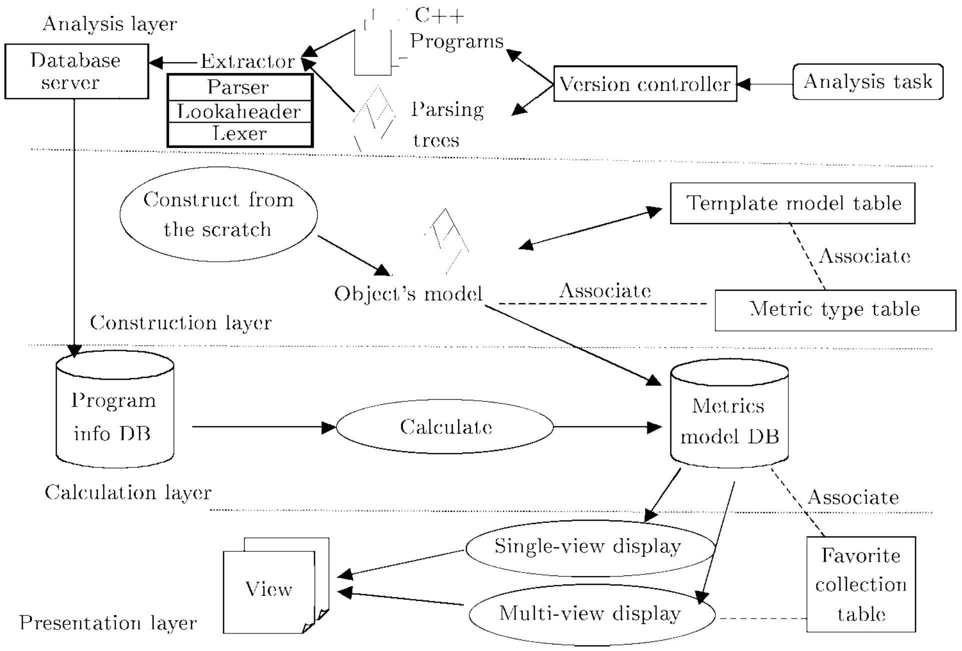

Fig.6. Layered architecture of JBOOMT.

• Construction Layer — Model Constructor

In the construction layer, metrics users can construct their hierarchical metrics models based on the implemented metrics from the scratch or by adapting existing models stored in the template model table. The bottom level of the model can be selected from a list of implemented metrics defined in the metrics type table. After users have finished constructing the desirable metrics model, the tool will save its specification in the metrics model database.

The metrics models are classified as method, class and system level metrics model based on the scope of the measured object. Each model is organized as a hierarchical diagram. The value of an upper level node is calculated based on the lower level node. The values of the bottom metrics level can be calculated from the information of the source code directly. An example of a metrics model shown in JBOOMT is illustrated in Fig.7. This model is a project level metrics model.

• Calculation Layer — Model Calculator

In this layer, according to the definition of the bottom level metrics of the specified model, the calculation part calculates the statistical values of an external node from the program information database by using the SQL language statement. After the measured values are attained, they should be normalized for comparison and computation together with other nodes. The parameters for absolute normalization such as the metrics type, "*a*" value and "*b*" value should be stored in the metrics type table. JBOOMT can normalize the measured values to normalized values between 0 and 1 according to the formula of the specified type and parameter values. The other way of normalizing the measured value is relative normalization, which is to divide the measured values by the pre-set criterion of the measured metrics values of a reference object. Therefore the criterion of compared metrics values should be stored in the database in advance. The normalized value can be obtained as the measured value is divided by the pre-set compared value.

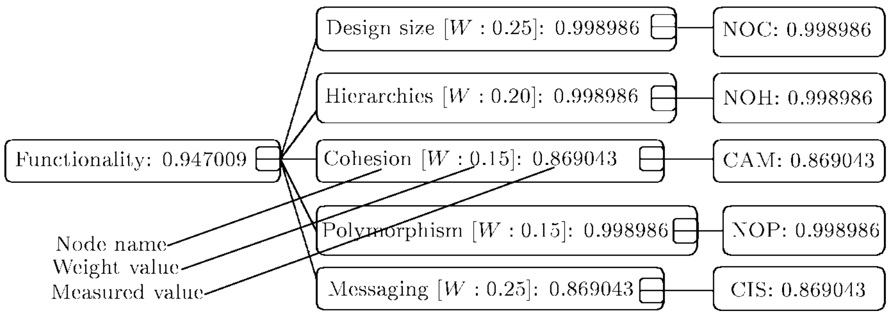
Functionality: 0.947009
Design size [W : 0.25]: 0.998986
Hierarchies [W : 0.20]: 0.998986
Cohesion [W : 0.15]: 0.869043
Polymorphism [W : 0.15]: 0.998986
Messaging [W : 0.25]: 0.869043
NOC: 0.998986
NOH: 0.998986
CAM: 0.869043
NOP: 0.998986
CIS: 0.869043
Node name
Weight value
Measured value

Fig.7. An example of metrics model showed in JBOOMT.

• Presentation Layer – Model Presenter

The values of each node in the hierarchical model can be displayed to metrics user. At the same time, the user can tailor the thresholds and default values of the metrics easily through the flexible user interface. The tool presents the results from different perspectives to facilitate the assessment of the program. To display the metrics values of the model effectively, the tool adopts tree diagram as the main representation form of metrics model.

## 4 Reusable Component Extraction Using the Approach

### 4.1 Applying JBOOMT in the Jade Bird System

As a software development environment which supports reuse, Jade Bird system is mainly to support the industrialization of software production based on component-architecture reuse and its kernel is a component library system — JBCL. The goal of JBCL is to describe, organize, store, manage and retrieve components to fulfil the requirements of a component-based reuse process. The JB Component Library Data Model is developed from the JB component model. Cooperating with other CASE tools, JBCL can support description (in JB Component Description Language — JB-CDL), classification, storage, and retrieval of components[7].

The management process of JBCL is indicated in Fig.8.

The reusable components in JB Component Library can be acquired by purposively development based on domain engineering, or by extraction from existing systems and the succedent reengineering. Of course, they may be purchased directly from a commercial software organization too. Before classifying and describing components, JBCL must qualify them to ensure that they have reached certain qualification standards. Only then can they be stored in JB Component Library. Users ask JB Component Library for components with JB Retrieval Tool and then composes them for new applications if these components meet the requirements. Before composing, some adaptation needs to be made if necessary.

There are several activities in the management process of JBCL to be assisted by JBOOMT. In the left part of Fig.8, Jade Bird Program Understanding Tool (JBPUS) together with JBOOMT can be used to aid user to understand the legacy systems.

Most importantly, the component extraction process involves identifying and qualifying reusable software in existing systems. A reusability metrics model can be formulated and then supported by JBOOMT. In this case, the type of normalization is set as absolute normalization. In this process, JBOOMT identifies those components such as classes from a large volume of source code whose metrics values of reusability model reach some specific threshold values. In this set of candidate components, users can utilize Jade Bird Reverse Engineering Tool (JBRET) to assist to reengineer the components. After the adaptation and improvement of the selected components in the reengineering process, components should be qualified before being inserted into the component library. And during the inserting process, the metrics information associated with the components may also be stored in the component library. This information can provide some valuable reference for users to decide which components can be retrieved when they are searching some candidate components. The qualification process also provides some feedback information to component extraction process to refine and improve the reusability metrics model.

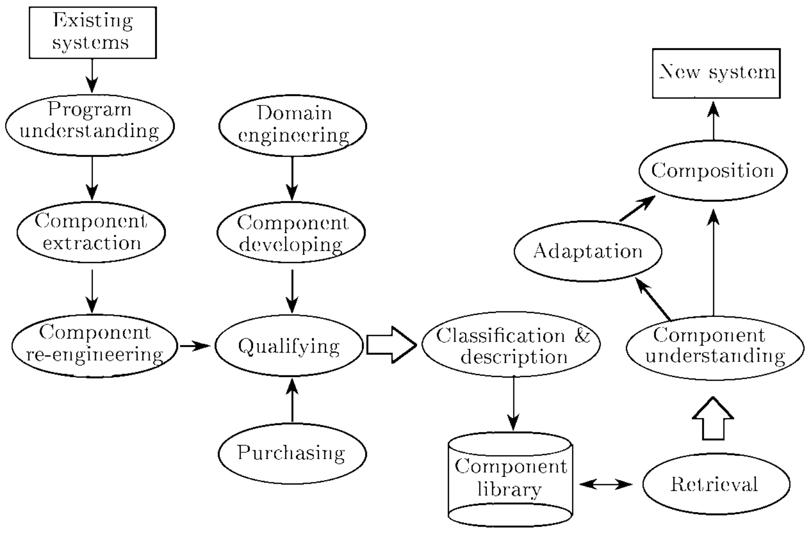

Fig.8. Management process of JBCL.

## 4.2 Applying JBOOMT in the Process of Domain Engineering

The objective of the domain implementation phase in domain engineering is to develop and organize the reusable information according to the domain model and Domain Specific Software Architecture (DSSA)[8]. This reusable information, which describes reusable components, can be obtained from legacy systems. One way to organize these reusable components is to insert them into the Jade Bird Component Library.

Reusable components should be engineered and designed to fit the domain model and DSSA. DSSA specifies the requirements of reusable components. In the process of domain implementation, it is generally required to design and code according to this requirement to produce the reusable components at the implementation level.

On the other hand, extracting the reusable components from existing systems is an effective way to acquire the domain components. According to the requirements of one specific component in DSSA, with the aid of Jade Bird Program Understanding System (JBPUS), domain experts can locate several candidate components, which have satisfied these requirements in several similar existing systems. Then JBOOMT is used to evaluate the reusability of these components according to the reusability metrics model customized by users. In this case, the way of normalization is set as relative normalization. Users can set one of these candidate components as the reference object. According to the result of evaluation in JBOOMT, users can select those components, which have most desirable results of the metrics model, to fit DSSA and then store them in Jade Bird Component Library.

## 4.3 Comparing Two Ways of Application of JBOOMT in CBSD Environment

In the above ways of applying JBOOMT to component-based software development environment, the objective of JBOOMT application to component library system is to identify the potential reusable components from a large group of components with different requirements or functions by using the absolute-normalized reusability metrics model in JBOOMT. It is usually hard to select a common reference object in a whole legacy system, so we have to adopt absolute normalization, although it is not satisfactory to derive a standard set of parameters for each metrics in the model.

However, it is necessary that any comparison among the metrics values of the quality attributes be done only among objects that have been developed to meet similar requirements and perform similar functions[5]. The objective of JBOOMT application to domain engineering is to identify the potential reusable components from a relatively small group of components with similar requirements and functions using the relative-normalized reusability metrics model in JBOOMT. The way of application also follows the principle mentioned above.

In both ways, when the selected components are inserted into component library, the reusability and quality metrics values of these components obtained from JBOOMT can also be stored in the Jade Bird Component Library to provide valuable information for the retrieval of components from component library.

## 5 Example

During the evolution process of JBOOMT, its database access interface component has been reengineered. We use the database access interface components before and after reengineering as the sample code which needs to be measured. Because these two components perform the same function, and only differ in implementation structure and details for some optimizations, the evaluation results of these two measured entities are comparable. To simplify the illustration, we call the component before reengineering the old system and the component after reengineering the new system.

JBOOMT facilitates users to create the model top-down. Users can first create the top node for the model and then create the child nodes for the existing nodes. By doing so, users can create the child nodes layer by layer till the nodes down to the bottom. Each node is classified as a metrics node or non-metrics node. Each internal node in the model is a non-metrics node whose value is calculated from the weighted values of those child nodes associated with this non-metrics node. And each external node or leaf node is a metrics node which is specified by certain metrics such as Line of Code or Number of Class, etc. And its value can be directly computed from the program information database which stores the program information extracted from source codes.

The other way to create the model is to reuse the template model stored in the template model database. After the template model has been loaded from the database, users can adapt it to a desirable one. Moreover, any metrics model can also be saved as a template model for future use.

In this example, we adopt two kinds of metrics models adapted from the QMOOD model[5]. One is the absolute metrics model and the other is the relative metrics model. Fig.9 shows the result of the absolute normalization model for the old system.

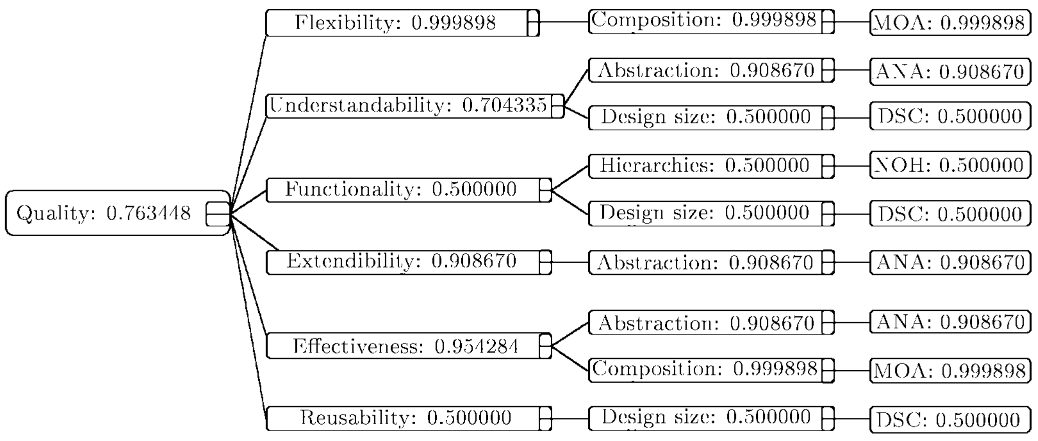

Fig.9. The result of the absolute normalization model for the old system.

For the new system, we also create the same metrics model for it. Fig.10 shows the result of the absolute normalization model for the new system.

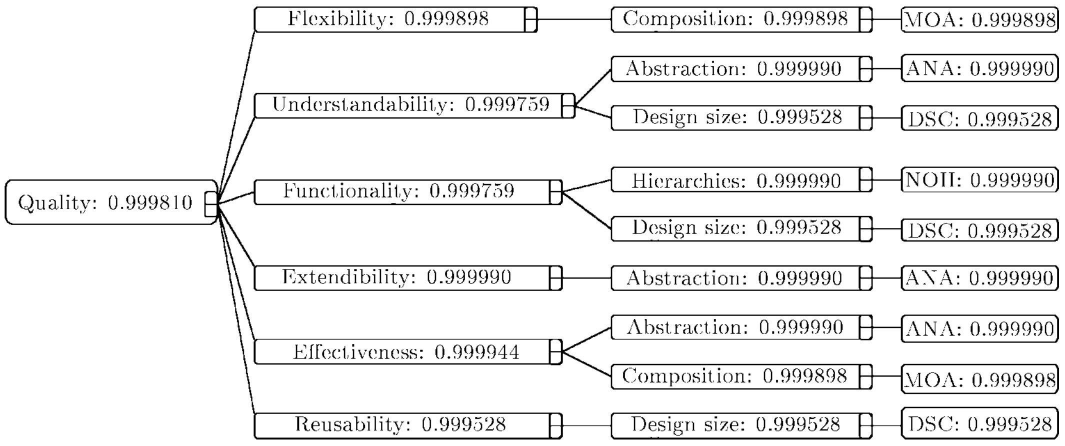

Fig.10. The result of the absolute normalization model for the new system.

The old system has been reengineered to produce the new system whose quality attributes have been improved. The metrics results of the two systems accord with the expectation. We construct another relative normalization metrics model with the old system as the comparison basis. The result is shown in Fig.11.

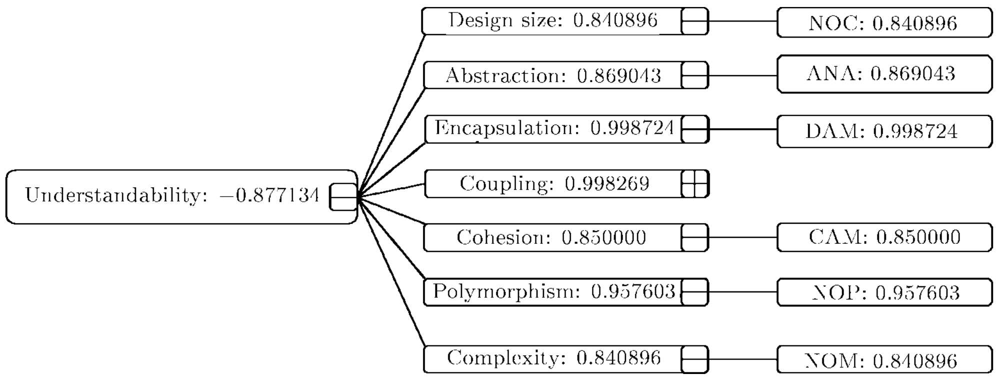

Fig.11. The result of the relative normalization model for the new system relative to the old system.

In this model, the baseline value of "Understandability" quality attribute is −1. Because the new system reduces the number of classes and association relationship, the new system becomes simpler and easier to understand. The result shows that the "Understandability" metrics value rises from −1 to about −0.877, which is consistent with the expectation.

## 6 Conclusion

In order to measure the quality attributes of the software products effectively, we propose a model-based approach to object-oriented software metrics. In this approach, users are guided to perform the measurement task according to five steps: the selection of the measured entities, the construction of the model, the normalization and computation of the model, the validation and comparison of the model, the improvement and application of the model. This approach proposes the top-down process to develop the metrics model and clarifies the absolute and relative normalization, which can be applied to specific situations respectively.

To implement this approach, we have designed an automated generic object-oriented metrics tool JBOOMT to support hierarchical models for the assessment of object-oriented software quality. The parser-based program analysis approach adopted by JBOOMT makes it possible for users to obtain the comprehensive and accurate program information to calculate metrics results. JBOOMT has been applied to the Jade Bird System and Jade Bird Domain Engineering method to support the software reuse in software development.

In the future, JBOOMT will be extended to support some other object-oriented languages such as Java. Moreover, it is also necessary to formulate some more appropriate metrics models for components in the component library and also some reusability metrics models which can be used to extract reusable components through the model-based approach. In addition, a lot of case studies should be done to validate and improve the correctness and usability of these metrics models. At the same time, the functionality of JBOOMT could be further enhanced.

## References

[1] McCall J A, Richards P G, Walters G F. Factors in Software Quality. Vols. I, Ⅱ, and Ⅲ (NTIS AD/A-049 014/015/055), Springfield: NTIS, 1977.

[2] Boehm B W, Brown J R, Kaspar H *et al.* Characteristics of Software Quality. Amsterdam: North-Holland, 1978.

[3] Fenton N E. Software measurement: A necessary scientific basis. *IEEE Trans. Software Engineering.,* March, 1994, 20(3): 199–206.

[4] Karlsson Even-Andre Chichester. Software Reuse: A Holistic Approach — Measuring the Effect of Reuse Chapter. New York: Wiley, 1995, pp.113–180.

[5] Bansiya J. A hierarchical model for quality assessment of object-oriented designs [dissertation]. Huntsville: University of Alabama in Huntsville, 1997.

[6] Yang Fuqing, Mei Hong, Yuan Wanghong *et al.* Experiences writing C++ compiler front end. *ACM SIGPLAN Notices,* 1998, 33(9): 95–102.

[7] Li Keqin, Guo Lifeng, Mei Hong, Yang Fuqing. An overview of JB (Jade Bird) component library system JBCL. In *Proc. 24th International Conference TOOLS Asia, Beijing,* Chen Jian, Li Mingshu, Christine Mingins, Bertrand Meyer (eds.), California: IEEE Computer Society Press, 1997, pp.261–267.

[8] James Petro, Michael E Fotta, David B Weisman. Model-based reuse repository — Concepts and experience. In *Proc. 7th International Workshop on CASE,* July 10–14, 1995, Toronto, Ontario, Canada, Hausi A Muller, Ronald J Norman (eds.), pp.60–69, IEEE Computer Society Press, Los Alamitos, California, USA.

**MEI Hong** is a professor in computer science at Peking University. He got his Ph.D. degree from Shanghai Jiaotong University in 1992. His research interests include software engineering and CASE tool, software reuse, software component technology, distributed object technology.

**YANG Fuqing** is a professor in computer science at Peking University. She is a member of the Chinese Academy of Sciences. She graduated from the Graduate School of Peking University in 1958. Her research interests include operating system, software engineering, CASE environment, software reuse and object-oriented technology.