Two-Phase Quality Measurement Model for Reusable Class Component

Sen-Tarng Lai
Department of Information Technology, National PingTung Institute of Commerce
51 Min Sheng E. Road, PingTung, 900 Taiwan
E-mail: stlai@npic.edu.tw

ABSTRACT

Software reuse is an important approach to increase software quality and productivity. In object-oriented software system, class component is a primitive element and a critical product. A class component with high reuse potential and high quality can to be a reusable class component. Domain experts and senior software engineers can identify the class component with high reuse potential. However, how to assure a class component with high quality becomes an important issue in software reuse. In this paper, static and dynamic metric data will be collected to measure the quality characteristics of class component. Modularity, complexity and document attributes are three major characteristics to affect the static quality of class component. Test completeness and performance evaluation are two major characteristics for the dynamic quality of class component. Individual software metric cannot measure the overall quality of the class component. Therefore, the software metrics must be combined, and conflicts among the software metrics must be reduced. For this, a Two-Phase Quality Measurement (*TPQM*) model that covers static view and dynamic view will be proposed in this paper. Applying this model, a highly flexible and practical metric combination approach can be created, the conflicts among individual primitive metrics can be reduced.

Keywords: reusable class component, software metrics, rule-based system, TPQM.

1. Introduction

There are many approaches to improve the software quality and productivity [3]. However, software reuse is one of most directly and efficiency approach. In [16], McClure

suggests several possibilities to be a software component such as program code, design specifications, plans, documentation, expertise and experience, and any information used to create software and software documentation. In [13], Langergan and Grasso discussed the design reuse and code reuse in software development. A code component in a reuse library is likely to be of little value; however, the detailed design documents should be very valuable for the adaptation to new applications. Thus, to be a suitable software component, Tracz [20] recommended that the detailed design documents should be associated with code modules. Class component is a primitive element of object-oriented software system that is produced by detailed design phase and accomplished by implementation phase. In this paper, the class component is regarded as the reusable component that includes class design specification, source code, and related documents.

Reuse potential and quality are two necessary conditions that make a class to be a Reusable Class Component (RCC). Domain expert and senior software engineer can identify the class component with high reuse potential. However, how to assure a class component with high quality becomes an important issue in class component reuse. Quality of class component not only has high relation to the class design, but also concerns the class implementation. There are several papers that discuss the hierarchical model of software quality [1, 8]. However, they put stress on the relationships between two quality characteristic levels, but do not investigate the relationships between quality characteristics and software metrics. Modularity, complexity and documents attributes are three major characteristics of design quality that can be measured by static metric data. Test coverage and performance evaluation are two major characteristics of implementation quality that can be measured by dynamic metric data. However, individual software metric cannot measure the overall quality characteristics of the class component. Therefore, the primitive metrics must be combined, and conflicts among the primitive metrics must be reduced. In this paper, static metric data will be collected and combined for measuring the design quality of class component and dynamic metric data will be collected and combined for measuring the implementation quality of class component. Based on the static and dynamic quality

measurement, a two-phase quality measurement model will be proposed.

In object-oriented design, there are many CASE (Computer Aided Software Engineering) tools to support the design methodologies that are proposed by the scholars or experts [4, 9]. The purpose of CASE tools is to improve design quality and productivity in object-oriented design. However, major quality characteristics of class component, which have high influence for the following phases, almost are neglected by the CASE tools. The class component without high quality may cause more development effort and cost of following phases and also cannot to be a potential RCC. In class design and implementation, several important quality characteristics will be fused to the class component. The quality characteristics of class component are depended on some primitive metrics. For measuring and controlling the quality of class component, in this paper, the primitive metrics collection will be surveyed and discussed then a quality measurement model will be proposed. In section 2, the detailed tasks of class component design and implementation will be identified and specified. Then, some criterions that determine the design and implementation quality of class component will be described, and some primitive metrics to measure the quality of class component will be discussed in section 3. The primitive metrics for different characteristics have different scale measurement values. In section 4, the static and dynamic primitive metric collection and normalization for metric combination will be described. Two-Phase Quality Measurement (*TPQM*) model for the quality of RCC will be proposed. In section 5, a rule-based system will be applied to the *TPQM* model for reducing conflict situations of metrics combination. Finally, a summary and our future work are given in the last section.

2. Class Design and Implementation

Life cycle of class component is class definition, class design, class implementation and class application. Class design and class implementation are two important steps to determine the usability and quality of class component.

2.1 Major Tasks of Class Design

Class component is a primitive element and key product in object-oriented software system. Class prototype is defined in class diagram of the object-oriented architecture design and class specification is designed in the class detailed design. Based on the principle of object-oriented design [4], five detailed design tasks for performing the class design can be described as follows:

(1) Class inheritance design: Inheritance is major feature in the object-oriented programming. For expressing the inheritance feature, in object-oriented design, the diagram of class hierarchy defines the inheritance relationship among the class components. Well-designed class inheritance can increase the productivity and maintainability, however, the misused inheritance relationship may cause high complexity and low flexibility.

(2) Class interface design: Like structured design [19], calling relation among the class components has to be defined in class component design. Message passing and calling hierarchy are two major tasks for performing the class interface design.

(3) Member functions and data members design: A class component is composed of several member functions and data members. For producing the high quality programming specification, it is a necessary task to clearly and correctly define and specify the member functions and data members of class component.

(4) Detailed logic structure design: After member functions have been defined, the detailed logic of member functions shall be designed. In this task, the detailed logic of member functions in class components shall be specified by PDL (Program Design Language) or pseudo code clearly.

(5) Detailed data structure design: After member data have been defined, the detailed data structure of member functions shall be designed. In this task, the detailed data structure of member functions in class components shall be specified by variable name, data type and storage space clearly.

The results of five class design tasks may affect the quality and operations of following

phases. The operations of implementation, testing and maintenance phases have tight relation with the tasks of class design. The contrastive relation between class design tasks and following phase operations is shown in Table 1. In class design, several quality characteristics which include modality, complexity and document attributes are fused to the class component.

2.2 Major Tasks of Class implementation

Pass through the class detailed design review, the development procedure will enter the class implementation phase. Class design specification should be transferred into the compactable source program, and the source program should be assured workable and correct in the class implementation phase. Based on the steps of class implementation, two implementation tasks for performing the class implementation can be described as follows:

(1) **Source code implementation:** According to class design specification and specific target language, the source program will be written and run on the target machine.

(2) **Class complement testing:** According to design specification, all kinds of test data should be generated and fed to the class component. Each test data has the specific object to find the hiding bugs and avoid the failure occurrence.

In class implementation, two quality characteristics which include test completeness and functional performance are fused to the class component.

3. Quality Characteristics of Class Component

In class design and implementation, several important quality characteristics, which will affect quality of class component, are fused to the class component.

3.1 Quality Characteristics of Class Design

The detailed design results of class component have high influence with the operations of following phases. Therefore, the quality of class component design becomes an important issue for the quality and productivity of overall software system. Several quality characteristics, which are modularity, complexity, document correctness, completeness, and

consistency, can be applied to measure the design quality of class component. Misused inheritance relation (for example multiple inheritance or more level inheritance), crude calling hierarchy design, and uncertain scope definition of member functions and data members may cause the class component with low modularity. The class component with low modularity not only may reduce the capability of extension and modification, but also may lose testability and productivity of following phases. In order to improve the maintainability, testability and productivity of following phases, the modularity of class component should be controlled effectively. Immature class logic structure and data structure design may cause the class component with high complexity. The class component with high complexity may always produce high error rates in the implementation phase and make low productivity in testing phase. For reducing the development cost and time in implementation and testing phases, complexity of class component should be reduced. Documents of class component have the objective to propagate the results of class design to the following phases. For continuing the results and design quality of class component, several quality characteristics of class component documents, which are correctness, completeness, and consistency, should be enhanced and controlled concretely.

3.2. Quality Characteristics of Class Implementation

Class implementation is a key step to transfer the class design specification into the source code. In order to verify the step of class implementation can satisfy the specification of class design, class testing becomes a necessary step to assure the result of class implementation. Function correctness and execution efficiency are two critical factors that affect the implementation quality of class component. Test completeness and performance evaluation are two major indications for measuring the implementation quality of class component. Hiding errors and implicit faults always make class component loss function correctness. Class component with incorrect function may cause the software system can not work normally. Class component with low performance may cause the software system can not meet requirement specification. Inefficiency instruction and not optimization source

code always make class component loss performance.

4. Two-Phase Quality Measurement Model

In this Section, static metric data will be combined for measuring the class design quality, dynamic metric data will be combined for measuring the class implementation quality and quality measurement model for RCC will be purposed.

4.1 Relationship between primitive metric and quality characteristic

Primitive software metrics for different quality characteristics has different measurement styles. For measuring the class design quality, the primitive metric data is called static metric that should be collected in class component no executing status. Modularity measurement can be combined with coupling metric and cohesion metric [7, 18, 19]. Class component with more level of inheritance or more source number of inheritance shall cause the class component with high coupling [7, 14, 18]. Class component with more level of calling relation or more source number of calling relation also shall cause the class component with high coupling. Inheritance and calling relation analysis tool for class component can help collect the coupling metric of class component. Analyzing the relative degree of all member functions in a specific class component can help collect the cohesion metric of class component. Complexity of a class component is depended on the logic structure, data structure and nesting depth level of program construct of member functions. For measuring the complexity of class component, McCable’s Cyclomatic complexity metrics [15], Halstead's Software Science [10] and program nesting level metric can be considered and collected. Documents attributes of class component has high influence with the quality and operations of following phases [6, 17]. Correctness, completeness and consistency are three major attributes to measure the documents quality of class component. In order to collect the metrics of correctness, completeness and consistency of class component documents, cautious document review and checklists of correctness, completeness and consistency should be applied to each document audit. The contrastive

relation among class design tasks, related quality metrics and affected operations is shown in Table 2.

For measuring the class implementation quality, the primitive metric data is called dynamic metric that should be collected in class component executing status. Test completeness is a major indicator for measuring the completeness of class testing. A class complement with high test completeness can also deduct this component with high function correctness. Statement coverage, branch coverage and member function coverage are three dynamic metrics for measuring the characteristic of class test completeness. Function performance is an important quality characteristic for the real-time or E-commerce software system. Best case, worst case and average case response time are three dynamic metrics for measuring the characteristic of class performance. In class component execution status, best case, worst case and average case response time of specific member functions can be collected. The contrastive relation among class implementation tasks, related quality metrics and affected operations is shown in Table 3.

4.2 Primitive metrics normalization

In the general case, a potential software quality characteristic is combined with several primitive metrics. Some primitive metrics, which are concerned with the quality characteristics of class component, have different scale measurement values in their representation. To combine these primitive metrics, which have different scale values in their representation, we recommend that all measure scale values of each primitive metric should be normalized to a value between 0 and 1. Close to 1 represents the most desirable value, and close to 0 represents the least desirable value. After normalization, the properties of correctness, objectivity, usability, and reliability for different primitive metrics should still be kept.

4.3 Two-phase quality measurement model

In the preceding section, static primitive metrics were identified to measure the design quality of class component and dynamic primitive metrics were identified to measure the implementation quality of class component. The primitive metrics play an individual role to measure the individual quality characteristic of class component. The individual measurements of quality characteristics cannot provide an overall picture of the quality measurement of class component. Thus, it leads to the consideration of combining them. Static primitive metrics must be combined for an overall static quality measurement of class component. Dynamic primitive metrics also have to be combined for an overall dynamic quality measurement of class component. For this, a metrics combination model that is based on the dynamically weighted linear combination is proposed. For measuring the static quality of class component, eight static primitive metrics are separated into three sets and combined as follows:

Set 1: Combine the cohesion and coupling metric into Modularity Measurement *(MM)* as Formula (1).

CPM: Metrics of Coupling W_p: Weight of CPM

CHM: Metrics of Cohesion W_h: Weight of CHM

$$MM = W_p * CPM + W_h * CHM \quad (1)$$

Set 2: Combine logic structure metric, data structure metric and nesting level of program construct into Complexity Measurement *(CM)* as Formula (2).

LSM: Metric of Logic Structure W_{ls}: Weight of LSM

DSM: Metric of Data Structure W_{ds}: Weight of DSM

NDM: Metric of Nesting Depth W_{nd} :Weight of NDM

$$CM = W_{ls} * LSM + W_{ds} * DSM + W_{nd} * NDM \quad (2)$$

Set 3: Combine the correctness, completeness, and consistency metrics into Documents Quality Measurement *(DQM)* as Formula (3).

C1M: Metrics of Correctness W_{c1}: Weight of C1M

C2M: Metrics of Completeness W_{c2}: Weight of C2M

C3M: Metrics of Consistency W_{c3}: Weight of C3M

$$DQM = W_{c1} * C1M + W_{c2} * C2M + W_{c3} * C3M \quad (3)$$

Then, three static high-level measurements *(MM, CM, DQM)* are combined into a static quality measurement of class component *(STQM)* as Formula (4).

MM: Modularity Measurement W_{mm}: Weight of MM

CM: Complexity Measurement W_{cm}: Weight of CM

DQM: Documents Quality Measurement W_{dqm}: Weight of DQM

$$STQM = W_{mm} * MM + W_{cm} * CM + W_{dqm} * DQM \quad (4)$$

For measuring the dynamic quality of class component, six dynamic primitive metrics are separated into two sets, and combined as follows:

Set 1: Combine the statement coverage, branch coverage and member function coverage into test completeness measurement as Formula (5).

STC: Statement Coverage W_{stc}: Weight of STC

BRC: Branch Coverage W_{brc}: Weight of BRC

MFC: Member Function Coverage W_{mfc}: Weight of MFC

$$TCM = W_{stc} * STC + W_{brc} * BRC + W_{mfc} * MFC \quad (5)$$

Set 2: Combine the best case, worst case and average case of response time into performance evaluation measurement of class component as Formula (6).

BTC: Best Case Response Time W_{btc}: Weight of BTC

WTC: Worst Case Response Time W_{wtc}: Weight of WTC

AVC: Average Case Response Time W_{avc}: Weight of AVC

$$PEM = W_{btc} * BTC + W_{wtc} * WTC + W_{avc} * AVC \quad (6)$$

Then, two dynamic high-level measurements *(CTM, PEM)* are combined into a dynamic quality measurement of class component *(DYQM)* as Formula (7).

TCM: Test Completeness Measurement W_{ctm}: Weight of CTM

PEM: Performance Evaluation Measurement W_{pem}: Weight of PEM

$$DYQM = W_{ctm} * TCM + W_{pem} * PEM \quad (7)$$

Finally, static quality measurement and dynamic quality measurement are combined into a quality measurement for RCC *(QMRCC)* as Formula (8).

STQM: Static Quality Measurement W_{stqm}: Weight of STQM

DYQM: Dynamic Quality Measurement W_{dyqm}: Weight of DYQM

$$QMRCC = W_{ctm} * CTM + W_{pem} * PEM \quad (8)$$

We call this measurement scheme a *Two-Phase Quality Measurement* (*TPQM*) Model (see Figure 1).

5. Rule-based metrics combination to reduce the conflicts

In *TPQM* Model, some conflict situations may occur in metrics or high level measurements combination, expertise and experience of object-oriented software engineering domain can help reduce the conflicts.

5.1 Conflict situations in metrics combination

According to the *TPQM* model, the conflict situations may occur in three formulas of primitive metrics combination as follows:

(1) There are two conflict situations between coupling and cohesion metrics of class component as follows:

- High coupling: If a class component has high coupling, then the influence of cohesion may be reduced. It is because high cohesion cannot increase the modularity of a class component that has high coupling.
- Low cohesion: If a class component has low cohesion, then the influence of coupling may be reduced. It is because low coupling cannot increase the modularity of a class component that has low cohesion.

(2) There are three conflict situations among complexity metrics as follows:

- High data structure complexity: If a class component has high data structure complexity, then the influence of depth of nesting and logic complexity may be reduced. It is because the depth of nesting and logic complexity can not reduce the high data structure complexity.
- High logic structure complexity: If a class component has high logic structure

complexity, then the influence of depth of nesting and data structure complexity may be reduced. It is because depth of nesting and data structure complexity can not reduce the high logic structure complexity.

- High logic and data structure complexity: If a class component has high logic and data structure complexity, then the influence of depth of nesting may be reduced. It is because the depth of nesting can not reduce the high logic and high data structure complexity.

(3) There are three conflict situations among test completeness measurement as follows:

- Low statement coverage: If a class component has low statement coverage in class test, then the influence of branch coverage and member function coverage may be reduced. It is because branch coverage and member function coverage can not increase the completeness of class test that has low statement coverage.
- Low branch coverage: If a class component has low branch coverage in class test, then the influence of statement coverage and member function coverage may be reduced. It is because statement coverage and member function coverage can not increase the completeness of class test that has low branch coverage.
- Low member function coverage: If a class component has low member function coverage in class test, then the influence of statement coverage and branch coverage may be reduced. It is because statement coverage and branch coverage can not increase the completeness of class test that has low member function coverage.

According to the *TPQM* model, the conflict situations may occur in a formula of high level measurements combination. In static quality measurement, there are three conflict situations among modularity measurement, complexity measurement and documents quality measurement as follows:

- Very low modularity measurement: If a class component has very low modularity measurement, then the influence of complexity measurement and documents quality measurement may be reduced. It is because a class component with

very low modularity measurement cannot become a high quality class component.

- Very low complexity measurement: If a class component has very low complexity measurement, then the influence of modularity measurement and documents quality measurement may be reduced. It is because a class component with very low complexity measurement cannot become a high quality class component.
- Very low documents quality measurement: If a class component has very low documents quality measurement, then the influence of modularity measurement and complexity measurement may be reduced. It is because a class component with very low documents quality measurement cannot become a high quality class component.

5.2 Rule-based metrics combination

For adjusting weight values to reduce conflict situations, the experience and knowledge of senior software engineers and domain experts should be acquired. Questionnaires, description of the preceding section, and relative formulas are provided to the senior software engineers and domain experts to collect the expertise. Based on the expertise and relative formulas, the production rules can be applied to the *TPQM* model. First, the production rules are applied to three formulas of primitive metrics combination for reducing the conflicts. According to Formula (1) and weight values, two production rules can be generated as follows:

$W_p = 0.55$, $W_h = 0.45$; *(set initial weight values)*

R_{11}: IF $CPM \leq 0.3$ *(high coupling)* **THEN** $W_p = 0.95$, $W_h = 0.05$;

R_{12}: IF $CHM \leq 0.3$ *(low cohesion)* **THEN** $W_p = 0.08$, $W_h = 0.92$;

According to Formula (2) and weight values, three production rules can be generated as follows:

$W_{ls} = 0.39$, $W_{ds} = 0.34$, $W_{nd} = 0.27$; *(set initial values)*

R_{21}: IF $LSM \le 0.3$ *(high logic structure complexity)*

THEN $W_{ls} = 0.91$, $W_{ds} = 0.06$, $W_{nd} = 0.03$;

R_{22}: IF $DSM \le 0.3$ *(high data structure complexity)*

THEN $W_{ls} = 0.07$, $W_{ds} = 0.88$, $W_{nd} = 0.05$;

R_{23}: IF $LSM \le 0.3$ *(high logic structure complexity)* and

$DSM \le 0.3$ *(high data structure complexity)*

THEN $W_{ls} = 0.51$, $W_{ds} = 0.44$, $W_{nd} = 0.05$;

According to Formula (5) and weight values, three production rules can be generated as follows:

$W_{stc} = 0.3$, $W_{brc} = 0.33$, $W_{mfc} = 0.37$; *(set initial values)*

R_{31}: IF $STC \le 0.3$ *(low statement coverage)*

THEN $W_{stc} = 0.91$, $W_{brc} = 0.03$, $W_{mfc} = 0.06$;

R_{32}: IF $BRC \le 0.3$ *(low branch coverage)*

THEN $W_{stc} = 0.02$, $W_{brc} = 0.94$, $W_{mfc} = 0.04$;

R_{33}: IF $MFC \le 0.3$ *(low member function coverage)*

THEN $W_{stc} = 0.02$, $W_{brc} = 0.03$, $W_{mfc} = 0.95$;

Second, the production rules are applied to a formula of high level measurements combination for reducing the conflicts. According to Formula (4) and weight values, three production rules can be generated as follows:

$W_{mm} = 0.33$, $W_{cm} = 0.32$, $W_{dqm} = 0.35$; *(set initial values)*

R_{41}: IF $MM \le 0.3$ *(very low modularity measurement)*

THEN $W_{mm} = 0.94$, $W_{cm} = 0.02$, $W_{dqm} = 0.04$;

R_{42}: IF $CM \le 0.3$ *(very low complexity measurement)*

THEN $W_{mm} = 0.04$, $W_{cm} = 0.90$, $W_{dqm} = 0.06$;

R_{43}: IF $DQM \le 0.3$ *(very low documents quality measurement)*

THEN $W_{mm} = 0.03$, $W_{cm} = 0.02$, $W_{dqm} = 0.95$;

High flexibility is a major feature of the production rules. Applying rule-based system to *TPQM* model can reduce conflict situations in metrics and measurements combination. In

addition, the rule-based system can isolate and identify the unqualified primitive metrics or unqualified quality characteristics that cause the low quality class component. Based on the clear evidences, the detailed defects should be found and the modification and adjustment plan should be proposed for improving the quality of class component.

6. Conclusions

Software reuse is an important approach to increase software quality and productivity. In object-oriented software system, class component is a primitive element and an important product. A class component with high reuse potential and high quality can be a RCC. For assuring class component quality, major tasks of class design and implementation are described in this paper. Based on the class design tasks, static metrics can be collected and quality characteristics can be measured for the design quality of class component. In this paper, the static quality characteristics of class component are separated into three sets as follows:

(1) Coupling and cohesion metrics for modularity measurement.

(2) Logic structure, data structure and nesting depth for complexity measurement.

(3) Correctness, completeness and consistency metrics for documents quality measurement.

Based on the class implementation tasks, dynamic metrics can be collected and quality characteristics can be measured for the implementation quality of class component. The dynamic quality characteristics of class component are separated into two sets as follows:

(1) Statement coverage, branch coverage and member functions coverage for test completeness measurement.

(2) Best cast, worst case and average case response time for performance measurement.

In addition, several primitive metrics for measuring the quality characteristics of class component are surveyed and discussed. The individual metric cannot measure the overall quality characteristic of class component. Therefore, the primitive metrics must be combined and conflict situations in metric combination should be reduced. In this paper, a

TPQM model was proposed. The model is based on the dynamically weight linear combination for primitive metrics, and applies the rule-based system to reduce the conflict situations in metrics combination. The advantage of *TPQM* model is to collect static and dynamic metric data for measuring the quality of RCC. High flexibility, high practicality, high adaptability and easy formulation are major features of *TPQM* model, and these features seem to be better than the other metric combination models [2, 5, 11, 12]. Several idea of *TPQM* model has been applied to software projects such as reusable component extraction and maintenance of service order processing system [11, 12]. Our feature work is to collect and analyze the feedback data to improve the *TPQM* model.

References

[1] Boehm, B. W., J. R. Brown, and M. Lipow, "Quantitative Evaluation of Software Quality", in Proceedings of the Secondary ICSE, pp. 592-605. (1976)

[2] Boehm, B. W., Software Engineering Economics, New Jersey: Prentice-Hall Inc. Pub. (1981)

[3] Boehm, B. W., M. H. Penedo, "A Software Development Environment for Improving Productivity," *IEEE Computer*, Vol. 17, No. 6, pp. 30-44 (1984)

[4] Booch, G., Object-Oriented Analysis and Design with Application, Menlo Park: Benjamin/ Cummings Inc. Pub. (1994).

[5] Conte, S.D., H.E. Dunsmore, and V.Y. Shen, Software Engineering Metrics and Models, Menlo Park: Benjamin/Cummings Inc. Pub. (1986)

[6] Deutsch, M. S. and R. R. Willis, Software Quality Engineering: A Total Technical and Management Approach, New Jersey: Prentice-Hall Inc. Pub. (1988)

[7] Fenton, N.E., Software Metrics - A Rigorous Approach, Chapman & Hall. (1991)

[8] Gillies, A.C.,Software Quality - Theory and Management, Chapman & Hall Inc. Pub. (1992)

[9] Graham, I., Object-Oriented Methods, Workingham, England: Addison-Wesley, Pub. (1991)

[10] Halstead, M.H., Elements of Software Science, New York: North-Holland Inc. Pub. (1977).

[11] Lai, S.T. and C.C. Yang, "A Software Metric Combination Model for Software Reuse," Proceedings of 1998 Asia-Pacific Software Engineering Conference (APSEC'98), pp. 70-77. (1998)

[12] Lai, S.T., "A Quality Measurement Model for Software Maintenance," Proceeding of the Second World Congress on Software Quality (2WCSQ), Sept.2000, Japan. (2000)

[13] Lanergan, R.G., and C.A. Grasso, "Software Engineering with Reusable Designs and Code," *IEEE Trans. Software Eng.*, Vol 10 No 5, pp.498-501. (1984)

[14] Lorenz, M. and J. Kidd, Object-Oriented Software Metrics: A practical Guide, New Jersey: Prentice-Hall Inc. Pub. (1994)

[15] McCabe, T.J., "A Complexity Measure," *IEEE Trans. Software Eng.*, Vol 2, No 4 pp.308-320. (1976)

[16] McClure, C.L., *The Three Rs of Software Automation: Re-engineering, Repository, Reusability,* Prentice Hall. (1992)

[17] Pressman, R.S., Software Engineering: A Practitioner's Approach, New York: McGraw-Hill Inc. Pub. (1993)

[18] Schach, S. R., Classical and Objec-Oriented Software Engineering With UML and Java, McGraw-Hill Companies (1999).

[19] Stevens, W., G. Myers, and L. Constantine, "Structured Design," IBM Systems Journal, vol. 13, no. 2, pp.115-139. (1974)

[20] Tracz, W., "Software Reuse Myths," *ACM SIGSOFT Software Emgineering Notes* Vol.3, (1), pp.17-21. (1988)

Table 1. A contrastive relation table for class design tasks and following phase operations

Related operations / Class design tasks	Implementation Phase	Testing Phase		Maintenance Phase	
		Class test	Integration (System) test	Extension	Modification
Inheritance design	✓		✓	✓	✓
Calling relation design	✓		✓	✓	✓
Member functions and data members design	✓	✓		✓	✓
Logic design	✓	✓		✓	✓
Data structure design	✓	✓		✓	✓

Table 2. A contrastive relation table for class design tasks, quality metrics and affected operations

Five Class Design Tasks \ Related Metrics & Operations	Related Quality Metrics	Affected Operations
Inheritance design	• Coupling Metric • Document Correctness • Document Completeness • Document Consistency	• Modification • Extension • Integration test
Calling relation design	• Coupling Metric • Document Correctness • Document Completeness • Document Consistency	• Modification • Extension • Integration test
Member functions and data members design	• Cohesion Metric • Document Correctness • Document Completeness • Document Consistency	• Modification • Implementation • Class testing
Detailed logic design	• Logic Structure Metric • Nesting Depth Metric • Document Correctness • Document Completeness • Document Consistency	• Implementation • Class testing • Revision
Detailed data structure design	• Data Structure Metric • Document Correctness • Document Completeness • Document Consistency	• Implementation • Class testing • Revision

Table 3. A contrastive relation table for class implementation tasks, quality metrics and affected operations

Class Implementation Tasks \ Related Metrics & Operations	Related Quality Metrics	Affected Operations
Source Code Implementation	• Data Structure Metric • Best Case Response Time • Worst Case Response Time • Average Case Response Time	• Modification • Extension • Revision • Integration test
Class Complement Testing	• Statement Coverage • Branch Coverage • Member Function Coverage • Document Correctness • Document Completeness • Document Consistency	• Modification • Extension • Revision • Integration test

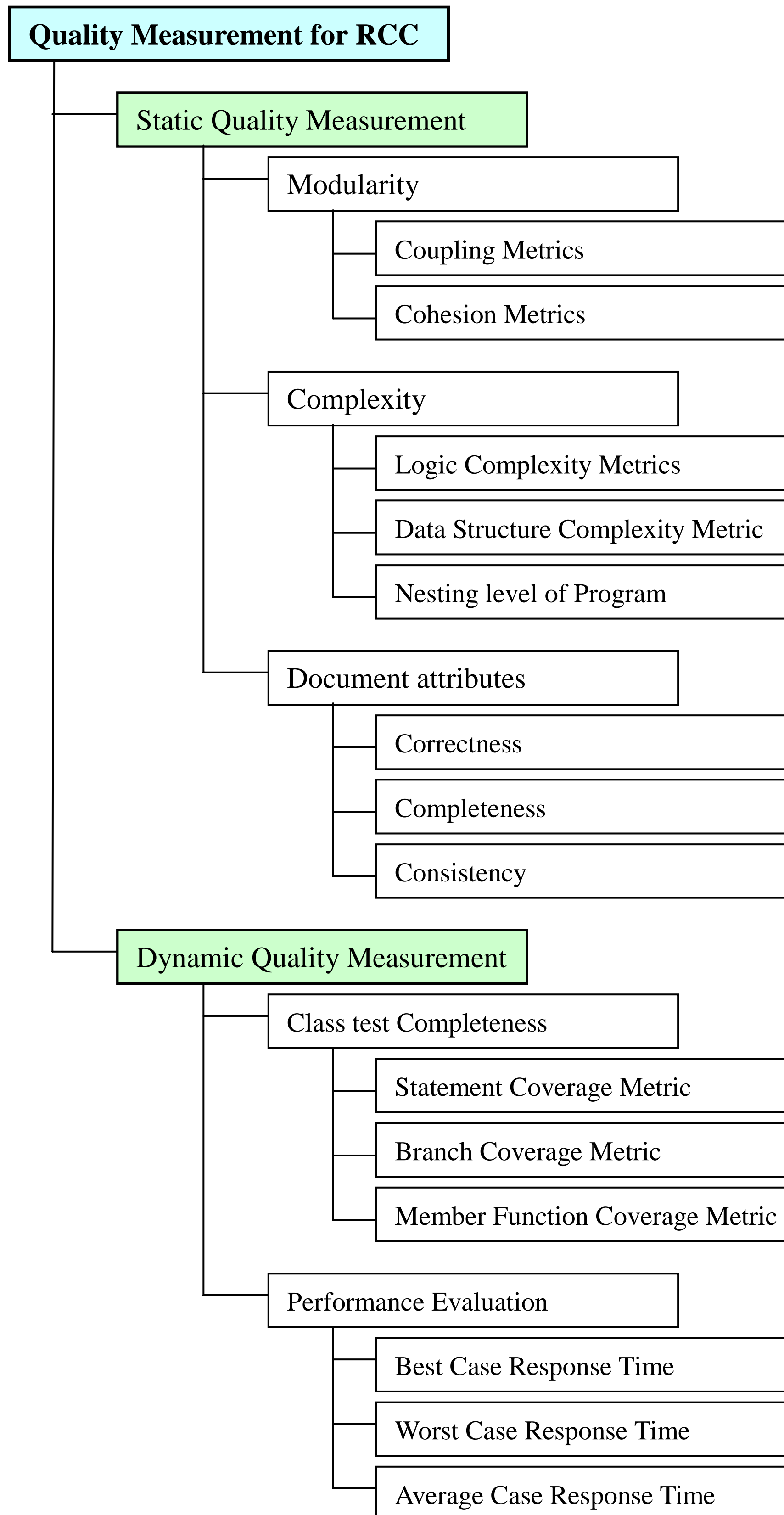

Figure 1. Two-Phase Quality Measurement Model for RCC