INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

SOFTWARE REUSE METRICS FOR COMPONENT QUALIFICATION IN REUSABLE VERIFICATION ENVIRONMENT

[1]Dr. V. Subedha, [2]Dr. S. Malathi
[1] Professor, Department of CSE, Panimalar Institute of Technology, Chennai, India
[2] Professors, Department of CSE, Panimalar Engineering College, Chennai, India
[1]subedha@gmail.com ,[2]malathi_raghu@hotmail.com

Abstract—Reusable software components must have concise abstractions so that reuser can efficiently locate, understand, compare and select the appropriate components from component reuse library. Some principles must be followed to ensure final components for reuse which should have good performance and high reuse value. Component qualification is qualifying the different components into different level and storing them in the component reuse library according to the quality of reusable software component. Deriving metrics for the assessing the reusability level of software components is a challenging task. Many metrics exist intuitively without mathematical models. Identifying all the parameters involved in reusability metric and the inter-relationships among those parameters is also a significant challenge. We describe a set of metrics to the assessment of component reusability. The aim of the proposed metrics is to predict the quality of software components and to classify the component into excellent candidate for reuse. We distinguish between a reuse metric, which measures the level of reuse within a project, system or organization and the reusability metrics, which measures the ability to employ an artefact in a context. We focus here on reusability measures.

.

Keywords - Reuse metrics; Component qualification; Reusability metrics; Software reusability; Reuse frequency

1. INTRODUCTION

Reusing software component is becoming a widely used approach to reduce the software development costs and shorten the software development production cycle. Software reuse can be applied to any life cycle product. Graaf et.al [1] identifies ten potentially reusable assets of software products as shown in the Table 1.

Table 1. Reusable Assets

Reference Architectures	Estimates
Source Code	Interfaces
Data	Plans
Design Patterns	Requirements
Documentation	Test Suite

As software engineering matures into a true engineering discipline, there is an increasing need for a corresponding maturity in repeatability, assessment, and measurement of the artifacts [2] associated with software. The real time users get advantage from reuse the existing software. To evaluate software and related assets for potential reuse, the software developers and adopters must share common measures.

Software reusability is an attribute that refers to the expected reuse potential of a software component. Software reuse not only improves productivity but also has a positive impact on the quality and maintainability of software products [3].

The aim of metrics is to predict the quality of the software products. The requirement today is to relate the reusability attributes with the metrics and to find how these metrics collectively determine the reusability of the software component.

The first step of the component qualification is to find out components offering the required functionality. However components retrieved from the reuse library is notoriously variable in quality. Consequently, we have to provide the re-user with an indication of quality components.

The identified five programs attributes for evaluating reusability are:

- Program Size
- Program Structure
- Program Documentation
- Programming Language
- Reuse Experience

The proposed metrics are used to evaluate the reusability of the components and it is evaluated in the terms for 'Reusable' or Non-Reusable components. The second section discusses about related work in this area. Third section gives the traditional metrics in general. Fourth section describes the factor and proposed metrics for component qualification for reusability in context level. In Last section conclusion is made.

I. RELATED WORK

One possible measure of a component's reusability comes from its success. There are basically two approaches to evaluate software reusability : qualitative and empirical . The qualitative methods require manual effort. On other hand, empirical methods depend on the objective data that can be collected automatically.

Cho et al [4] proposes a set of metrics for measuring various aspects of software components like complexity, customizability and reusability. The work considers two approached to measure the reusability of the component. In first approach the reusability is measured in the design phase in a component development process and in second approach particular component's reuse level per application in a component based software development is measured.

Washizaki et al [5] discusses the importance of reusability of components in order to realize the reuse components efficiently and propose a component reusability model from the view point of component reuser.

Richard W. Selby [6] identified factors that characterize module reuse are low coupling, high cohesion and discussed that CK metric suit is able to target all the essential attributes of OO-based software.

Husein and Oxley [7] describe coupling and cohesion metrics suite is presented to evaluate object-oriented software. These metrics may be applied to assess reusability.

Parvinder Singh and Sandhu and Hardeep Singh, [8] [9] have used metric based approach for identifying a software module and the reusability was obtained with the help of Fuzzy Logic and Neuro-Fuzzy. This research shows how metrics can be used to identify the quality of a software component.

Himani Goel and Gurbhej Singh [10] proposed an Expectation Maximization based clustering approach to evaluate the reusability prediction of function based software systems. Here, the metric based approach is used for prediction. The metrics used for measuring the reusability of software modules are Cyclometric complexity, volume, regularity, reuse frequency and coupling.

Sonia Manhas et al [11] proposed Reusability Evaluation model for procedure based software systems to calculate reusability value which enables to identify a good quality component for reuse. The framework of metrics proposed for this model are Cyclometric complexity, volume, regularity, reuse frequency and coupling. Neural Network techniques are explored to design the reusability evaluation model.

Parvinder & Shalini [12] proposed particle swarm optimization technique along with the four variants of conjugate gradient algorithm to train the feed forward network. The performance of the train neural networks is tested to evaluate the reusability level of the procedure based software systems.

Ajay Kumar [13] propose Classification of the reusability of software components using Support Vector Machine and also the identification of reusable software modules in Procedure Oriented System is based on software metrics like Cyclometric complexity, Volume, Regularity, Coupling and Reuse frequency.

In all existing model the static metrics are defined to evaluate the quality of the components for reusability where we propose dynamic metrics to evaluate the quality of the components for reusability.

II. TRADITIONAL METRICS

Reusability can be measured directly or indirectly. Second, there are two types of reusability metrics – potential and actual. In this section we deal with some important traditional metrics to measure the reusability.

A. Metric Suit for Function Oriented Paradigm

The set of traditional metrics which were able to target all the essential attributes of function-based software is mentioned by Selby [6] in his latest findings, so we refer the metrics set for Function Oriented Paradigm are as follows:

- Cyclometric Complexity [9][14]
- Halstead Software Science Indicator [9] [14]
- Regularity Metric [9][14]
- Reuse-Frequency Metric [9][14]
- Coupling Metric [9]

Cyclometric Complexity Using Mc Cabe's Measure: According to Mc Cabe, the value of Cyclometric Complexity (CC) can be obtained using the following equation (1) :

CC = Number of predicated nodes + 1 (1)

(1)

Where predicate nodes are the directed graph, made for the components, where the decisions are made. If the complexity is low then reuse of component will not repay the cost. Otherwise high value of complexity indicates poor quality, high development cost, low readability, poor testability and prone to errors i.e. high rate of failure. Hence, the value of Cyclometric Complexity of a software component should be in between upper and lower bounds as an contribution towards reusability [15] [16].

Halstead Software Science Indicator: According to this metric volume of the source code of the software component is expressed in the following equation (2) :

$$\text{Volume} = (N1 + N2) \log_2 (n1+n2) \qquad (2)$$

Where, n1 is the number of distinct operators that appear in the program, n2 is the number of distinct operands that appear in the program, N1 is the total number of operator occurrences. N2 is the total number of operand occurrences. If the volume is high means that software component needs more maintenance cost, correctness cost and modification cost [17]. On the other hand, less volume increases the extraction cost, identification cost from the repository and packaging cost of the component. So the volume of the reusable component should be in between the two extremes.

Regularity Metric: Regularity is the ratio of estimated length to the actual length Regularity is to predict length based on some regularity assumptions. As actual length (N) is sum of N1 and N2. The estimated length is shown in the following equation (3):

$$\text{Estimated Length} = n1 \log_2 n1 + n2 \log_2 n2 \qquad (3)$$

The closeness of the estimate is a measure of the regularity of Component coding is calculated as in the equation (4)

$$\text{Regularity} = 1-\{(N-N')/N\} = N'/N \qquad (4)$$

High value of Regularity indicates the high readability, low modification cost and no redundancy of the component implementation [17]. Hence, there should be some minimum level of Regularity of the component to indicate the reusability of that component.

Reuse Frequency Metric:"Reuse frequency" is the measure of function usefulness of a component [6]. Reuse frequency is calculated by comparing number of static calls addressed to a component with number of calls addressed to the component whose reusability is to be measured. Reuse-frequency is calculated as in the equation (5):

$$\text{Reuse Frequency} = \frac{n(C)}{\frac{1}{n}\sum_{i=1}^{n} n(Si)} \qquad (5)$$

Where n(C) is total number of reference to the Component, n(Si) is total number of reference for each Standard Components in the existing environment & n is the total number of component in the existing environment. Hence, there should be some minimum value of "Reuse Frequency" to make software component really reusable.

Coupling Metric: As coupling increases, there is decrease in understandability and maintainability, so there should be some maximum value of the coupling associated with a software component, beyond which the component becomes non-reusable [14] [15].

B. *Metric Suit for Object Oriented Paradigm*

Reusability evaluation system for Object oriented Software components metric suit are given below:

- Weighted methods per class (WMC) [18][19]
- Depth of inheritance tree (DIT) [18][19]
- Number of Children (NOC) [18][19]
- Coupling Between Object Classes (CBO) [18][19]
- Lack of Cohesion in Methods (LCOM) [18][19]

Weighted Methods per Class (WMC): According to this metric if a Class C, has n methods and C1, C2 …, Cn be the complexity of the methods, then WMC(C) = C1 + C2 +… + Cn. Mc Cabe's complexity metric is chosen for calculating the complexity values of the methods of a class; the value is normalized so that nominal complexity for a method takes on a value of 0 to 1. If all method complexities are considered to be unity, then WMC = n i.e. the number of methods existing in that class [18] [19].

Depth of Inheritance Tree (DIT): According to this metric Depth of inheritance of a class is the maximum length from the node to the root of the tree. More is the depth of the inheritance tree greater the reusability of the class corresponding to the root of that tree as the class properties are shared by more derived classes under that class. Greater depth dilutes the abstraction and there is a need to set the minimum and maximum DIT value for a class as a contribution towards the reusability [18][19]. The definition of DIT is ambiguous when multiple inheritance and multiple roots are present as the alternative length of the path is not being considered in case of multiple inheritance. If all the ancestor classes coming in common path are added to the ancestor classes of alternative paths then that will be the true representation of the theoretical basis of the DIT metric.

Number of Children (NOC): According to this metric, Number of children (NOC) of a class is the number of immediate sub-classes subordinated to a class in the class hierarchy. Thus, greater is the value of NOC, greater will be the reusability of the parent class. Hence, there should be some minimum value of NOC for a parent class for its reusability [18][19]. Theoretical basis of NOC metric relates to the notion of scope of properties. It is a measure of how many sub-classes are going to inherit the methods of the parent class. The definition of NOC metric gives the distorted view of the system as it counts only the immediate sub-classes instead of all the descendants of the class. The NOC value of a class (say class '*i*') should reflect all the subclasses that share the properties of that class as shown in the following equation (6):

$$NOC(i) = N + \sum_{i}^{Allsubclasses} NOC(i) \qquad (6)$$

Where *N* is the total number of immediate subclasses of class *i*.

Coupling between Object Classes (CBO): According to this metric, "Coupling between Object Classes" (CBO) for a class is a count of the number of other classes to which it is coupled. Theoretical basis of CBO relates to the notion that an object is coupled to another object if one of them acts on the other.

Here, we are restricting the unidirectional use of methods or instance variables of another object by the object of the class whose reusability is to be measured. As Coupling between Object classes increases, reusability decreases and it becomes

harder to modify and test the software system. So, there is a need to set some maximum value of coupling level for its reusability and if the value of CBO for a class is beyond that maximum value then the class is said to be non-reusable [18][19].

Lack of Cohesion in Methods (LCOM): Consider a Class C1 with *n* methods M1, M2 ..., Mn. Let $\{I_j\}$ = set of instance variables used by method Mi .There are *n* such sets $\{I_1\}$, $\{I_2\}$, ..., $\{I_n\}$. Let P = $\{(I_i , I_j) \mid I_i \cap I_j = \phi\}$ and Q

= {(I_i , I_j) | $I_i \cap I_j \neq \phi$ }. If all n sets {I1}, {I2}, {In} are ϕ then P = ϕ . Lack of Cohesion in Methods (LCOM) of a class can be defined as:

LCOM = |P| - |Q|, if |P| > |Q|
LCOM = 0 otherwise

The high value of LCOM indicates that the methods in the class are not really related to each other and vice versa. It means that low value of LCOM depicts high internal strength of the class which results into high reusability. So, there should be some maximum value of LCOM after that class becomes non-reusable [18] [19].

III. PROPOSED METRIC SUITE FOR COMPONENT QUALIFICATION

Our motivation of this section is identifying suitable metrics with lower and upper bounds to support the qualification of the components as good or bad candidate for reusability.

Four primitive metrics are selected to measure the three factors. Table 2 contains the definitions of the metrics used for measuring the coverage analysis, time and functional usefulness.

Table 2. Definitions of Metrics to measures the factors

Metrics	Definition
Statement coverage	This metric reports whether each executable statement is encountered
Branch Coverage	This metric reports whether Boolean expressions tested in control structures
Extraction Time	This metric is used to measure the Extraction time of each component
Reuse Frequency	This metric is an indirect measure of the functional usefulness of a component

A. Three Factors for Component Qualification in context level

Measuring coverage analysis report: Using the functional specification the re-user generates, executes and associates with component a set of test cases and functional coverage report of the component is collected. Three commonly used measures of coverage driven functional verification are statement coverage, branch coverage and logical path coverage. We use the statement coverage and branch coverage as metrics for coverage analysis to qualify the candidate component for reusability.

Measuring time: The components having the extraction time less than the average extraction time is qualified for reuse. The reason for choosing the extraction time as metrics is to speed up the process of reuse. The extraction time and the optimal path for the extraction is calculated using a scheme called as minimum Extraction Time First.

Measuring functional Usefulness: The reuse frequency is an indirect measure of the functional usefulness of a component. We measure the functional usefulness that frequently used system is a good candidate for reuse in context level in similar domain. Hence we choose the metrics reuse frequency as a qualifier for classifying the components. Reuse frequency of each component can be calculated using the equation (7).

$$\text{Reuse Frequency} = \frac{n(C)}{\frac{1}{n}\sum_{i=1}^{n} n(Si)} \qquad (7)$$

Where n(C) is total number of reference to the Component, n(Si) is total number of reference for each Standard Components in the existing environment & n is the total number of component in the existing environment

B. Four Metrics for measuring the reusability in context level

Functional coverage report: The functional coverage report consists of statement coverage and branch coverage. The cut-offs for qualifying the component based on statement coverage is 100% and the for branch coverage it is divided into three level 85-90% as LOW, 90-95% as MEDIUM and greater than 95% as HIGH.

Reuse Frequency: The reuse frequency is an indirect measure of the functional usefulness of a component. We measure the functional usefulness that frequently used system is an excellent candidate for reuse in context level in similar domain. Hence we choose the metrics reuse frequency as a qualifier for classifying the components.

Minimum Extraction time: The components having the extraction time less than the average extraction time is qualified for reuse. The extraction time of each component is calculated using METF method. The reason for choosing the extraction time as metrics is to speed up the process of reuse.

C. Metric suit for measuring overall reusability

Reuse-Utility-Level: Reuse-Utility-Level is the most important reuse metrics and this metric is very simple to measure. It is the ratio of No. of Reused software components to the No. of software components available in the existing environment. The reuse utility level is measured using the equation (8).

$$\text{Reuse UtilityLevel} = \frac{n(RSC)}{n(SC)} * 100 \qquad (8)$$

Where n(RSC) is total number of Reusable Software Components & n(SC) is total number of Standard Components existing environment

Component Reuse percentage: This is the metric used to measures how much components are qualified for reusing from identified set and it is given as in equation (9)

$$\text{Component Reuse Percentage} = \frac{n(Q)}{n(Q)+n(NQ)} * 100\% \qquad (9)$$

Where n(Q) is the number of components qualified for reuse and n(NQ) is the number of components not qualified for reuse.

Precision: Precision for the qualification is the number of true positives divided by the total number of elements labeled as positive class and false class. Precision is calculated with equation (10) as follows.

$$\text{Precision} = TP / (TP+FP) \qquad (10)$$

Where True Positives (TP) is components correctly classified as Reusable Components and False Positives (FP) refer to Non-reusable modules incorrectly labeled as Reusable components.

Recall: Recall in this context is defined as the number of True Positives divided by the total number of elements that actually belong to the positive class (i.e. the sum of true positives and false negatives). The recall can be calculated as follows by the equation (11).

$$\text{Recall} = TP / (TP + FN) \quad (11)$$

Where True Positives (TP) are components correctly classified as Reusable Components and False Negatives (FN) refer to Reusable modules incorrectly classified as Non-Reusable Components.

Accuracy: The accuracy is the percentage of the predicted values that match with the expected values of the reusability for the given data. The best system is that having the High Accuracy, High Precision and high Recall value.

IV. CONCLUSION

Assessing the value of reuse is a major concern in the software industry. For assessing the value we should measure it by using metrics and models. A metric is a quantitative indicator of an attribute. Generally we use metric based approach for assessment because of its success in evaluation of reusability. So, in this paper we propose a set of metrics to measure the reusability. Our work involves identifying the reusability assessment metrics. It is found that the complexity is the most important factor in deciding the better reusability of function oriented software. In case of object oriented software, Coupling and complexity collectively play significant role in high reusability. In reusability based on context level the factors involved are functional coverage analysis report, extraction time and reuse frequency. The best condition of all factors helps obtain high reusability degree of software component. The proposed metrics can be used for procedure oriented and object oriented component. Our work contributes to the knowledge base. All the metrics value is stored in the dynamic metrics library for each reuser for further reusability. The future work can be extended in following directions

- Other dimensions of quality of software can be considered for mapping the relation of attributes.
- The metrics proposed here can be used in future for further study and empirical validation of these metrics for component based on context level.

Our immediate next work to be done is to apply the proposed metrics on large scale systems to identify the benefits and limitation.

REFERENCES

[1] B. Graaf, M. Lormans and H. Toetend, "Embedded Software Engineering: The state of the Practice ", IEEE Software, Vol. 20, No. 6, pp.61-69, Nov 2003.

[2] Z.M. Zhang, Y.T. Zhuang, Y.H. Pan, "Object-Oriented Software Reverse Engineering," *J. of Computer Research and Development*, vol.40, no.7, pp.1062–1068, 2003.

[3] Gill, Nasib S., "Importance of Software Component Characterization for Better Software Reusability", ACM SIGSOFT Software Engineering Notes, vol. 31 No. 1, Jan 2006, pp. 1-3.

[4] Eun Sook Cho, Min Sun Kim, Soo Dong Kim, "Components Metrics to Measure Component Quality", Proceedings of the Eighth Asia-Pacific Software Engineering, pp. 419 - 426, 2001

[5] Hironori Washizaki, Hirokazu Yamamoto and Yoshiaki Fukazawa, "Software Component Metrics and It's Experimental Evaluation", Proceedings of the International Symposium on Empirical Software Engineering (ISESE 2002), Oct 2002.

[6] Richard W. Selby, "Enabling Reuse-Based Software Development of Large-Scale Systems," IEEE Transaction of Software Engineering, Vol. 31, No. 6, PP. 495-510, Jun 2005

[7] Husein. S and A. Oxley, "A Coupling and Cohesion Metrics suite for Object-Oriented Software", Proceedings of International Conference on Computer Technology and Development, Malaysia, pp.421-425, Nov 2009

[8] Parvinder Singh Sandhu and Hardeep Singh, "A Fuzzy-Inference System Based Approach for the Prediction of Quality of Reusable Software Components", International Conference on Advanced Computing and Communications, ADCOM 2008. on page(s): 349 – 352

[9] Parvinder Singh Sandhu and Hardeep Singh, "Automatic Reusability Appraisal of Software Components using Neuro-Fuzzy Approach", International Journal Of Information Technology, vol. 3, no. 3, 2006, pp. 209-214.

[10] Himani Goel and Gurbhej Singh, "Evaluation of Expectation Maximization based Clustering Approach for reusability Prediction of Function based Software Systems", International Journal of Computer Applications,Vol 8,No. 13,pp.13-20,Oct 2010.

[11] Sonia Manhas, Rajeev Vashisht, Parvinder S. Sandhu and Nirvair Neeru,"Reusability Evaluation model for Procedure Based Software Systems", International Journal of Computer and Electrical Engineering",vol.2.,No.6, pp.1107-1111,Dec 2010.

[12] Parvinder S. Sandhu and Shalini Chhabra,"A Comparative Analysis of Conjugate Gradient Algorithms & PSO Based Neural Network Approaches for Reusability Evaluation of procedure Based software Systems", Chiang Mai j. Sci, Vol 38(special issue), pp.123-135, Jan 2011

[13] Ajay Kumar, "Measuring Software Reusability using SVM based Classifier Approach", International Journal of Information Technology and Knowledge Management.", Vol. 5, No. 1, pp.205-209, Jan 2012

[14] T. MaCabe, "A Software Complexity measure", IEEE Trans. Software Eng., vol. SE-2 (December 1976), pp. 308-320.

[15] G. Caldiera and V. R. Basili, Identifying and Qualifying Reusable Software components, IEEE Computer, (1991), pp. 61-70.

[16] Challagulla, V.U.B., Bastani, F.B., I-Ling Yen, Paul, (2005), "Empirical assessment of machine learning based software defect prediction techniques", 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, WORDS 2005, Feb 2005, pp. 263-270.

[17] Frakes, W.B. and Kyo Kang (2005) "Software Reuse Research: Statusand Future", IEEE Trans. Software Engineering, vol. 31, issue 7, July 2005, pp. 529 - 536.

[18] Chidamber, S.R. and Kemerer, C.F., " A Metric Suite for Object Oriented Design", IEEE Trans. Software Engineering , Vol. 20, pp. 476- 493,1994.

[19] Parvinder Singh and Hardeep Singh (2005) "Critical Suggestive Evaluation of CK METRIC", Proc. of 9th Pacific Asia Conference on Information Technology (PACIS-2005), Bangkok, Thailand, July 7 – 10, 2005, pp 234-241.