Reusability Metrics for Object-Oriented System: An Alternative Approach

Parul Gandhi gandhi2110@gmail.com
Department of Computer Science & Business Administration
Manav Rachna International University
Faridabad, 121001, India

Pradeep Kumar Bhatia pk_bhatia2002@yahoo.com
Department of Computer Science & Engineering
G. J. Universityof Science and Technology
Hisar, 125001, India

Abstract

Object-oriented metrics plays an import role in ensuring the desired quality and have widely been applied to practical software projects. The benefits of object-oriented software development increasing leading to development of new measurement techniques. Assessing the reusability is more and more of a necessity. Reusability is the key element to reduce the cost and improve the quality of the software. Generic programming helps us to achieve the concept of reusability through C++ Templates which helps in developing reusable software modules and also identify effectiveness of this reuse strategy. The advantage of defining metrics for templates is the possibility to measure the reusability of software component and to identify the most effective reuse strategy. The need for such metrics is particularly useful when an organization is adopting a new technology, for which established practices have yet to be developed. Many researchers have done research on reusability metrics [2, 9, 3, 4]. In this paper we have proposed four new independent metrics Number of Template Children (NTC), Depth of Template Tree (DTT) Method Template Inheritance Factor (MTIF) and Attribute Template Inheritance Factor (ATIF), to measure the reusability for object-oriented systems.

Keywords: Object-Oriented Metrics, Reusability, Generic Programming, Template, Inheritance.

1. INTRODUCTION

Reuse of existing software components increase the quality and productivity in software development and maintenance. Software reuse reduces the amount of software that needs to be produced from scratch and hence less testing time for new software. Industrial observers suggest that a reuse strategy could save up to 20% of development costs [10]. With reuse, software development becomes a capital investment. C++ templates are used to support the concept of reusability in object-oriented programming. An object-oriented software system is a collection of classes which are abstract data types and templates are a way of making classes more abstract without actually knowing what data type will be handled by the operations of the class. The ability to have a single class that can handle several different data types means the code is easier to maintain, and it makes classes more reusable. This raises questions about how generic programming included in the form of templates in the code can be measured to identify effectiveness of this reuse strategy. The measurement of reuse would help developers to monitor current levels of reuse and provide insight in developing software that is easily reused. In this paper, four metrics are proposed to measure amount of reusability included in the form of templates. Reuse can be classified in one of the following ways: public/private, verbatim /generic/leveraged, and direct/indirect. Public reuse is reuse of externally constructed software while private reuse is reuse of software within a product [14]. We will continue our study based on object-oriented metrics suite given by Chidamber and Kamerer[5] and the MOOD metric suite[7]. Among the Chidamber and Kamerer metric suite the most significant metrics for reusability are Depth of Inheritance Tree (DIT), which indicates the length of inheritance [8] and Number of Children (NOC) which indicates the width. Method Inheritance factor (MIF) and Attribute Inheritance factor also indicates the reusability [7, 4]. Our main focus of this work is on using these metrics to evaluate the characteristic of template classes which helps in developing reusable software.

2. OO METRICS- AN OVERVIEW

OO Metrics play an important role in ensuring the desired quality and productivity of the software project. To ensure the quality of the OO software many researchers have proposed metrics suite [9, 5, 6]. The main metrics for OO software are briefly described below [5].

Metric 1: Weighted Method Per Class (WMC) is the number of methods defined in class. WMC is used to predict time and effort required to develop and maintain the class. Classes with many methods are more application specific which limits the possibility of reuse It can be defined as referred from[5]

$$WMC = \sum_{i=1}^{n} C_i$$

Where

- n No of methods defined in a class
- C_i Complexity of method i

Metric 2: Depth of Inheritance tree (DIT) can be defined as maximum inheritance path from the class to the root class. The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex. As a positive factor, deep trees increase reusability because of inheritance feature[5].

Metric 3: Number of Children (NOC) can be defined as number of immediate sub-classes of a class. NOC measures the breadth of a class hierarchy. Higher the value of NOC , fewer the faults, which is desirable[5]. NOC, therefore, primarily evaluates efficiency, reusability, and testability [8, 12,13].

Metric 4: Coupling Between Object (CBO) can be defined as number of classes to which a class is coupled. Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class [5].

Metric 5: Response for a Class (RFC) can be defined as set of methods that can potentially be executed in response to a message received by an object of that class. Large RFC indicate more faults. Classes with high RFC become more complex and their testing become more complicated [5].

Metric 6: Lack of Cohesion of Methods (LCOM) indicates the lack of cohesion of methods. *Given n* methods $M_1, M_2, ..., M_n$ contained in a class C_1 which also contains a set of instance variables $\{I_i\}$. Then for any method M_i we can define the partitioned set of [5]

$$P = \{(I_i, I_j) \mid I_i \cap I_j = \varphi\} \text{ and } Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \varphi\}$$
$$\text{then LCOM} = |P| - |Q|, \text{ if } |P| > |Q|$$
$$=0 \text{ otherwise}$$

3. GENERIC PROGRAMMING WITH TEMPLATE - AN OVERVIEW

Templates are useful feature of object- oriented programming to implement generic constructs which can be used with any arbitrary type. Templates can be used to create a family of classes and functions. C++ templates provide a way to re-use source code as opposed to inheritance. With the help of template a single class can be used to handle different types of data and a single function can be used to accept different types of data which makes the code easier to maintain and classes more reusable. templates provide sufficient information to a compiler's optimizers (especially the inliner) to generate code that is optimal in both time and space[1]. Templates can be classified into two categories: Class Templates and Function Templates.

3.1 Class Template

Class Templates allow the classes to operate with generic type. These classes are generic type and member function of these classes can operate on different data types. This will overcome the limitation of classes to hold objects of any particular data type. The following class template shown in Figure 1 illustrates how the complier handles creation of objects using class template:

```
template <class T1, class t2>
class Sample
{
T1 a;
T2 b;
………
};
/*when objects of templates class are created using the
following statements Sample <int, float>s*/
/*The compiler creates the following class sample with two
data members one is of int type and other is of float type*/
class sample
{
int a;
float b;
………
};
```

FIGURE 1: Source Code for Class Template

3.2 Function Template

The C++ language [11] supports parameterized types and functions in the form of templates. With the help of templates the programmer can declare group of functions that works for all data types. Function Templates compactly and conveniently perform identical operations for each type of data compactly and conveniently. Based on the argument types provided in calls to the function, the compiler automatically instantiates separate object code functions to handle each type of call appropriately. The main advantage of generic function is that they overcome the limitation of general function which operates only on a particular data type. The following Function template shown in Figure2 illustrates how the complier handles creation of functions using function template:

```
template <class T>
T large (T a, T b}
{
return a > b ? a : b ;
}
// Function can be called as follows
int a,c=5,d=4;
a=large(c,d);
/*The compiler creates the following
function for data type int:*/
int large (int a, int b)
{
return a > b ? a : b ;
}
```

FIGURE 2: Source Code for Function Templates

4. PROPOSED METRICS

In this section we have proposed four independent metrics and illustrate their use by computing their values on example source code.

4.1. Metric 1: Number of Template Children (NTC)

The metric NTC can be defined as number of immediate sub-classes of a Template class.

```
template<class T>
class A
{
……
};
template<class S>
class B: public A<S>
{
……..
};
Class c
{
……
};
```

FIGURE 3: Source code for calculating metric NTC

In this example there is one class B which inherits from a template class A therefore Number of Template Children (NTC) is 1. The more the value of metric Number of Template Children (NTC), more reusable software components are included in the projects.

4.2. Metric 2: Depth of Template Tree (DTT)

The metric DTT can be defined as maximum inheritance path from the class to the root template class. In this example class B inherits from class A and class C inherits from class B Thus if we start the root node at level 0 the Depth of Template Tree (DTT) will be 2.

```
template<class T>
class A
{.....}
class B: public A
{....};
Class C :public B
{…..};
Class D
{…..}
```

FIGURE 4: Source code for calculating metric DTT

The greater the metric Depth of Template Tree (DTT) value, greater is the reusability since generic programming is form of reuse.

4.3. Metric 3: Method Template Inheritance Factor (MTIF)

MTIF is defined as the ratio of the sum of the methods inherited from template classes of the system under consideration to the total number of available methods (locally defined plus inherited) for all classes.

$$\text{MTIF}=\frac{\sum_{i=1}^{n} M_t(C_i)}{\sum_{i=1}^{n} M_a(C_i)} * \text{NO}$$

n Total number of classes

NO Number of Objects of Template classes

M_iC_i Number of methods declared in class i

M_tC_i Number of the methods inherited from template class i

$M_a(C_i)$ $M_iC_i + M_tC_i$ Total no of methods invoked

```
template <class T>
class A
{
T large (T a, T b)
{
……
}
};
class B: public A<S>
{
S sum(S c,S d)
{
….
}
};
```

FIGURE 5: Source code for calculating metric MTIF

No of template inherited method =0+1 =1
No of methods declared in each class 1+1 =2
Total=3
If we create two objects of class B

MTIF=(1*2)/3=0.6

The greater the value of metric Method Template Inheritance Factor (MTIF) will result in the increased code reusability.

4.4. Metric 4: Attribute Template Inheritance Factor (ATIF)

AIF is defined as the ratio of the sum of attributes inherited from template classes of the system under consideration to the total number of available attributes (locally defined plus inherited) for all classes.

$$\text{ATIF}=\frac{\sum_{i=1}^{n} A_t(C_i)}{\sum_{i=1}^{n} A_a(C_i)} * \text{NO}$$

n	Total number of classes
NO	Number of Objects of Template class
A_iC_i	Number of attributes declared in class i
A_tC_i	Number of the attributes inherited from template class i
$A_a(C_i)$	A_iC_i + A_tC_i Total no of attributes accessed

```
template<class T>
class A
{
T a;
T b;
};

template<class S>
class B: public A<S>
{
S c;
S d;
};
```

FIGURE 6: Source code for calculating metric ATIF

No of template inherited attributes =0+ 2 =2
No of attributes declared in each class 2+2 =4
Total= 6
If we create two objects of class B

ATIF= (2*2)/6=0.6

The more the value of metric Attribute Template Inheritance Factor (ATIF), more will be the code reusability.

5. DISCUSSION

The amount of function and class templates included in the system is dependent on the number of generic constructs needed in the application. The more the value of proposed metrics, more reusable software components are included in the projects. Table1 shows the effect of these four independent metrics on Reusability. The function and class templates are easier to maintain but difficult to test. The observations made by analyzing metrics are shown below:

- The greater the metrics value, greater would be the code reusability.
- The greater the metrics value, less redundancy is involved in coding
- Experimental results show that the proposed metrics are only applicable for template classes.
- If there are large number of function and class templates in a system, the testing and debugging of the function and class becomes more complicated since it requires greater level of understanding at the part of developer.

Name of metrics	Relative value of metrics	Implication for Reusability
NTC	increases	increases
DTT	increases	increases
MTIF	increases	increases
ATIF	increases	increases

TABLE 1: Relative values for metrics ideal for Reusability Quality factor

6. CONCLUSION

As organizations implement systematic software reuse programs in an effort to improve productivity and quality, they must be able to measure their progress and identify the most effective reuse strategies. In this paper, we have proposes set of four object- oriented metrics and results are analyzed based on object-oriented code. Our work presented in this paper emphasis to proactively use template mechanism which aid to analyze how much reusability is incorporated in the coding process and hence increases the use of generic programming. The advantage of using these metrics is the added insight gained about reduction in lines of code via using more generic programming in the form of function and class templates. The metrics presented in this paper have been found to be very useful to find the extent of reusability included in the code in the form of class and function templates. The main objective of our work explores the main aspect of software reusability. The most obvious extension of this work is to analyze the degree to which these metrics correlate with managerial performance indicators such as testing, maintenance effort and quality.

7. REFERENCES

[1] Douglas Gregor, Jaakko Jarvi, Jeremy Siek, Bjarne Stroustrup, Gabriel Dos Reis Andrew Lumsdaine Concepts: Linguistic Support for Generic Programming in C++ OOPSLA, 2006.

[2] J. Barnard, 'A New Reusability Metric for Object-Oriented Software', Software Quality Journal, Vol 7 no. 1, 1998.

[3] K.K.Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra Empirical Study of Object-Oriented Metrics, JOURNAL OF OBJECT TECHNOLOGY Vol. 5, no. 8, 2006.

[4] R.Harrison, S.J.Counsell, and R.V.Nithi, “An Evaluation of MOOD set of Object-Oriented Software Metrics”, IEEE Trans. Software Engineering, Vol. SE-24, no.6, June, 1998, pp. 491-496.

[5] S. R. *Chidamber* , C. F. Kemerer, A Metrics Suite for Object Oriented Design, *IEEE Transactions* on Software Engineering, Vol .20 no..6, June 1994, pp. 476-492.

[6] W.Frakes and C.Terry, “Software Reuse: Metrics and Models”, ACM Computing Surveys, Vol. 28, no.2, June 1996, pp. 415-435

[7] F.B. Abreu,”The Mood Metric Set”, Proc. ECOOP’95 Workshop on Metrics, 1995.

[8] J.Bieman and S.Karunanithi, “Candidate reuse metrics for Object Oriented and Ada Software”, In Proceedings of IEEE-CS First International Software Metrics Symposium.

[9] K.K.Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra “Software Reuse Metrics for Object-Oriented Systems”, In Proceedings of ACIS Third International conference on Software Engineering Research, Management and Applications, 2005.

[10] B.Henderson-sellers, Object-Oriented Metrics, Measures of Complexity, rentice Hall,1996.

[11] Bjarne Stroustrup. The C++ Programming Language. Addison-Wesley, special edition, 2000.

[12] Lorenz, Mark and Jeff Kidd, Object-Oriented Software Metrics, Prentice-Hall, Englewood Cliffs, N.J., 1994.

[13] McCabe & Associates, McCabe Object-Oriented Tool User's Instructions, user manual, 1994.

[14] N.E.Fenton, Software Metrics, A rigorous approach.Chapman and Hall, New York, 1991.

.