Analytical Analysis of Generic Reusability: Weyuker's Properties

Parul Gandhi[1], Pradeep Kumar Bhatia[2]

**[1] Department of Computer Science & Business Administration,
Manav Rachna International University
Faridabad, 121001, India**

**2 Department of Computer Science & Engineering
G. J. University of Science and Technology
Hisar, 125001, India**

Abstract

Reusability is the key concept in today's software development environment. The concept of reusability can be achieved by Generic programming approach. C++ templates help us to develop generic code which results in reusable software modules and also identify effectiveness of this reuse strategy. Many researchers have already developed various reusability metrics [9] [7]. In this paper we emphasis on evaluating reusability metrics on weyuker's set of properties. Weyuker's list of properties has always been a point of reference and suggested as a guiding tool in identification of a good complexity measure by several researchers. We have chosen some recently reported reusability metrics Method Template Inheritance Factor (MTIF) and Attribute Template Inheritance factor (ATIF) and evaluated them against Weyuker's set of principles. We divide our work in a two-step framework. In the first step the metrics are analytically evaluated against a formal list of Weyuker's properties and in the second step we calculate LOC metric value by using three different programs designed using template and inheritance features of object-oriented programming and observe that by using template with inheritance property we can reduce number of lines of a project to a great extent.

Keywords: *Reusability, Weyuker's Properties, Object-Oriented metrics, Generic Construct.*

1. Introduction

Code Reusability is the significant benefit of object-oriented programming that plays an important role in improving and enhancing the quality of software. C++ templates strengthen this concept of generic reusability by developing software reusable module [10] such as function template and class template.

In the last decade various appropriate ways has been designed to measure the complexity of program [3, 2]. Weyuker developed a formal list of properties for software metrics [3] and has evaluated a number of existing software metrics using these properties Weyuker's [11] proposed these properties to evaluate complexity measure when only traditional programming languages were in use. These properties were also used by some researchers for evaluation of popular object oriented metrics for example Chidamber's metrics [2] and Kapsu's metric [5], although the object-oriented features are entirely different in nature. It is not mandatory that all types of metrics satisfy all the weyuker's properties. Our approach is to figure out, for a given property, which type of the metrics will not satisfy Weyuker's property. In this paper, we have chosen two recently reported generic reusability metrics [9] and then analytically analyzed these metrics against Weyuker's proposed set of nine axioms. Further, we design three sample programs by keeping in mind the features of object-oriented programming language and then calculate LOC metric value of these programs which shows that the amount of lines of code (LOC) of the projects designed using template feature can be reduced to a great extent as compare to one designed without template.

We organize the paper in following sections: in the first section reusability metrics are defined which we want to evaluate against weyuker's properties. Section 3 and 4 states weyuker's properties and evaluate the defined reusability metrics against these properties. The discussion made is presented in section 5. The last section constitute conclusion.

2. Reusability Metrics

We have chosen two recently reported reusability metrics which we want to evaluate against weyuker's properties. These metrics are [9]:

Metric 1: 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.

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

n	Total number of classes
NO	Number of Objects of Template classes
MiCi	Number of methods declared in class i
MtCi	Number of the methods inherited from template class i
Ma (Ci)	MiCi + MtCi Total no of methods invoked

Metric 2: Attribute Template Inheritance Factor (ATIF)
ATIF 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.

$$\text{ATIF}=\frac{\sum_{i=1}^{n} A_t(C_i)}{\sum_{i=1}^{n} A_a(C_i)} * \text{NO} \qquad (2)$$

n	Total number of classes
NO	Number of Objects of Template class
AiCi	Number of attributes declared in class i
AtCi	Number of the attributes inherited from template class i
Aa (Ci)	AiCi + AtCi Total no of attributes accessed

3. Weyuker's Properties

Weyuker's properties [3] designed to evaluate software complexity measures. Several researchers have recommended various properties that software metrics should posses to enhance their usability. Weyuker's properties are not without criticism as many authors have criticized [8] [4] this approach but still it give an important basis to classify a complexity measure. Weyuker's properties states that [1].
Let μ be metric of program A and B:

Property 1:
Given a class A another class B can always be found such that, μ (A) ≠ μ (B). This implies that not every class can have the same value for a metric.

Property 2:
Let c be a nonnegative number. Then there are finite numbers of program with metric c such that μ (A) = c.

Property 3:
There are distinct programs A and B such that, μ (A) = μ (B).

Property 4:
There exist programs A and B having same functionality but their complexities could be different.
$(\exists A)(\exists B)(A \equiv B \ \& \ (\mu (A) \neq \mu (B))$.

Property 5:
Weyuker's fifth property is the property of monotonic. When two programs are concatenated, their metric value should be greater than the metrics of each of the individuals.
$(\forall A)(\forall B) (\mu (A) \leq \mu (A+B)) \ \& \ \mu (B) \leq \mu (A+B))$

Property 6:
This property suggests non-equivalence of interaction. If there are two program bodies of equal metric value which, when separately concatenated to a same third program, yield program of different metric value. For programs A, B & C
$(\exists A)(\exists B)(\exists C)(\mu (A) = \mu (B) \ \& \ \mu (A+C) \neq \mu (B+C))$.

Property 7:
This property is not applicable for object oriented metrics [2].

Property 8:
It specifies that "if A is a renaming of B, then μ (A) = μ (B).

Property 9:
This property states that the sum of the metric values of a program could be less than the metric value of the program when considered as a whole
$(\exists A)(\exists B) (\mu (A) + \mu (B) < \mu (A+B))$.

4. Evaluation of MTIF and ATIF on Weyuker's Properties

This section analyzes the applicability of weyuker's properties for two object-oriented reusability metrics Method Template Inheritance Factor (MTIF) and Attribute Template Inheritance factor (ATIF) [9]. Weyuker's first four properties are general in nature and assumed to be satisfied by any sensible measure.

Property 1 to Property 4:

Let μ (A) and μ (B) represent the metric value of programs A and B respectively. As two object-oriented programs can always differ in template value thus the metrics defined above satisfies the first property of weyuker's list. Metrics taken in this paper can have many different values, it satisfies second property. If there are two different unrelated programs implemented using the concept of template, then the metric value of both the programs can be same as it is nothing but the summation of methods or attributes inherited from template class. The choice of generic attributes or methods is a design decision which does not depend upon the functionality of software, this satisfies Property 4.

Property 5:
Let μ (A) = a and μ (B) = b Then μ (A+B) = a+b-α
Where α is the number of common functionality between program A and B. Thus it is possible that
(∀A)(∀B) (μ (A) ≤ μ (A+B)) & μ (B) ≤ μ (A+B))
Therefore, Property 5 is not satisfied.

Property 6:
Now, let A and B be two programs such that u (A) = u (B) = a and let C be another program such that u(C) = c.
Then, μ (A+C) = a+c-α and
μ (B+C) = a+c-β
It shows that for programs A, B & C μ (A+C) ≠ μ (B+C)), this satisfies Property 6.

Property 8:
The renaming of systems A and B does not affect the metric values. Hence, Property 8 is satisfied.

Property 9:
For any two programs A and B Let μ (A) = a and μ (B) = b then μ (A+B) = a+b-α
Thus a+b ≥ a+b-α, therefore property 9 is not satisfied.

5. Discussion

We apply these metrics to three different programs written using object-oriented language and conclude that by using the concept of template with inheritance property of object-oriented programming we can minimize the redundant code in our project to a great extent shown in table1 and hence increase the reusability of the software.

Table 1 LOC for three Sample Programs

Programs	***LOC$_t$***	***LOC$_{wt}$***
Program 1	34	43
Program 2	52	72
Program 3	38	54

Where

LOC$_t$ is the Line Of Code value for the program written using template
LOC$_{wt}$ is the Line Of Code value for the program written without using template

We also observe that lines of code (LOC) metric value were reduced when templates with inheritance property were used. A line of code is any line of programming text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the lines. This specifically included all lines containing program header, declarations, and executable and non-executable statements [6]. This is the predominant definition for LOC used by researchers. It also decreases the efforts required for implementation of the programs with an acceptable accuracy. The amount of reduction in LOC with the use of templates with inheritance is shown in Figure 1. The amount of LOC increased in projects, which do not include templates is shown in Figure 2. The increase in LOC varies from 26-42%.

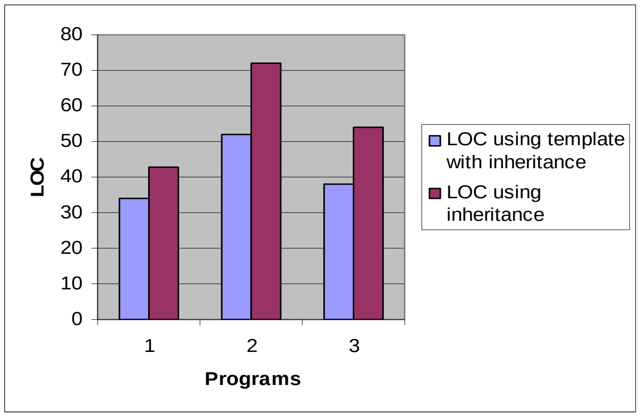

Figure 1: Comparison of LOC in Project 1 to Project 3

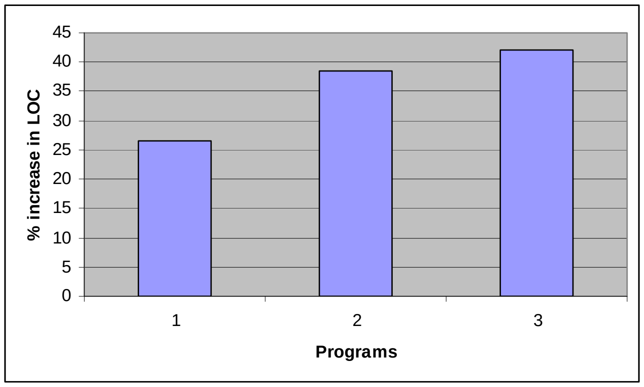

Figure 2: Percentage increase in LOC in projects not using templates

Our research work emphasis on the use of generic programming approach and also advice managers to proactively use generic programming opportunities in the development of object-oriented modules. Some important points/observations regarding the Weyuker's properties, and object oriented metrics are also suggested.

- An object oriented metric should satisfy Weyuker's properties 1, 2, 3, 4, 6 and 8.
- The observations say that proof of properties 5, 7 and 9 varies from metric to metric.
- In OOP, a class is an abstraction of the problem space, and the order of statements within the class definition has no impact on eventual execution or use .This is the reason that most of the OO metrics should not satisfy property 7.
- Weyuker's Property 9 has received a mixed response regarding its applicability to object oriented software metrics

6. Conclusions

Although Weyuker's properties are not without criticism but still gaining popularity as an evaluation criterion for OO measures. Weyuker's proposed these properties to evaluate complexity measure for procedural languages but now a days it also plays an important role to evaluate complexity measure for object-oriented languages. In this paper we have taken two recently reported reusability metrics Method Template Inheritance Factor (MTIF) and Attribute Template Inheritance factor (ATIF) and analyze them on the basis of weyuker's set of properties. One more objective of our research is to reduce the line of code by making use of generic construct. The above metrics helps in improving the reusability of software by avoiding redundant code and hence results in reduction in LOC. 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.

References

[1] B.Henderson-sellers, Object-Oriented Metrics, Measures of Complexity, Prentice Hall, 1996.
[2] Chidamber, S.R and Kemerer, C.F.:A Metric Suite for Object Oriented Design,IEEE Transactions on Software Engineering, Vol. 20, No. 6, 1994, pp. 476-493.
[3] E. J. Weyuker. Evaluating software complexity measures. IEEE Trans. Software Engineering, Vol. 14, no. 9, 1988, pp. 1357–1365.
[4] H.Zuse, Software Complexity: Measures and Methods, Walter de Gruyter, Berlin, 1990.

[5] Kapsu K., Shin, Y.,Chisu W.: Complexity Measures for Object-Oriented Program Based on the Entropy, Software Engineering Conference, 1995. Proceedings, 1995 Asia Pacific. 1995, pp.127 – 136.
[6] K.K Aggarwal, Yogesh Singh, Software Engineering, New Age International Publishers, 2001.
[7] 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.
[8] N.E.Fenton, Software Metrics, A rigorous approach. Chapman and Hall, New York, 1991.
[9] Parul Gandhi & Pradeep Kumar Bhatia Estimation of Generic Reusability for Object Oriented Software An Empirical Approach", ACM SIGSOFT Software Engineering Notes Vol. 30, no 3, 2011, pp. 1-4.
[10]Parul Gandhi & Pradeep Kumar Bhatia," Reusability Metrics for Object-Oriented System: An Alternative Approach" International Journal of Software Engineering (IJSE), Vol. 1, No 4, 2010, pp. 63-72.
[11]Stockhome, S.G., Todd, A.R., Robinson, G.A.,: A Framework for Software Quality Measurement, IEEE Journal on Selected Areas in Communications,Vol.8, No.2, 1990, pp.224-233.

Acknowledgments

I express my sincere gratitude and acknowledgement towards Dr. Pradeep Kumar Bhatia, Associate Professor, who guided me. It was his constant support and inspiration without which my efforts would not have taken this shape. I sincerely thank him for this, and seek his support for all my future endeavors.

Parul Gandhi is pursuing PhD (computer science).She received her MPhil (computer science) degree in 2008. She holds an MCA from MDU, Rohtak. She is currently working as Lecturer with Manav Rachna International University, Faridabad, FBC Department. Her area of specialization includes Software Engineering, Software quality improvement.

Dr Pradeep Kumar Bhatia Ph.D. (Computer Science & Engineering), May 2005 from Guru Jambheshwar University of Sci & Tech, HISAR Presently working as Associate Professor with Guru Jambheshwar University Hisar. His current areas of specialization include Software Engineering and Software Quality Improvement. He holds to his credit around 19 Years of experience of teaching and research. He has more than 50 research papers in various reputed journals and also written various books in various fields.