Dynamic inheritance coupling metric-design and analysis for assessing reusability

Neha Gehlot and Jagdeep Kaur*

CSE Department,
ITM University,
Gurgaon, Haryana, India
Email: neha12sep009@itmindia.edu
Email: jagdeep@itmindia.edu
*Corresponding author

Abstract: Moving along the different phases of software development life cycle measuring the quality of software has always been a challenging task but is considered immensely useful. The area of software metrics, especially pertaining to object-oriented software system, has expertise in describing the characteristics of a software system for the past few decades. Software metrics numerically extract relationships among given components in a software system and relate those measurements to the system's quality. Thus, software metrics predict the current level of software quality and in turn initiate a feedback process that may lead to further improvement of a software system. But the actual behaviour of the software can only be measured from information collected at runtime. Thus, there is a need of evolving such software metrics that are based on the runtime analysis of a software system. These metrics are known as dynamic metrics. In this paper, new dynamic coupling and inheritance metrics for object-oriented systems is designed and validated on a metric tool developed in Java measuring inheritance coupling, complexity, class independent factor and relating the calculated measures to a software quality attribute-reusability which is useful in software quality assessment.

Keywords: metrics; quality attribute; inheritance; coupling; reusability.

Reference to this paper should be made as follows: Gehlot, N. and Kaur, J. (2015) 'Dynamic inheritance coupling metric-design and analysis for assessing reusability', *Int. J. Software Engineering, Technology and Applications*, Vol. 1, No. 1, pp.118–133.

Biographical notes: Neha Gehlot completed her MTech in Software Engineering in 2014 from ITM University, India. She completed her BTech in Computer Science and Engineering from Maharishi Dyanand University, Rohtak, India in 2012.

Jagdeep Kaur is an Assistant Professor in CSE Department at ITM University, India. She is pursuing her PhD from Gautam Bhuddha University, Greater Noida, India in Software Engineering (2012). She completed her MTech in CSE from Punjabi University, Patiala, India in 2005 and BTech in CSE from Punjab Technical University, Jalandhar, India in 2003.

1 Introduction

In order to enhance the quality of software system metrics are considered as one of the most important tools used in software engineering. They are concerned with measurement in software engineering. Software metrics can be defined as means to supply meaningful and timely management information, together with the use of measurement-based techniques to improve process and its products in software development. The software metric concept in which measurement-based techniques are applied to processes, products and services to supply useful engineering information metrics (Briand et al., 1999).

Many object-oriented software metrics have evolved over the years. Most of these metrics extract useful information from a static software code. The related information can be number of lines of code, number of classes in the code, coupling among classes, etc. Such an analysis of static code is called static analysis of software code and the metrics that carry out such analysis are called static metrics. But the actual behaviour of the software can only be measured from information collected at runtime. Thus, there is a need of evolving such software metrics that are based on the runtime analysis of a software system. These metrics are known as dynamic metrics.

Many quality attributes are available like complexity, coupling, reusability and many metrics are also present to measure them (Arisholm et al., 2004). To calculate the degree of dependency between the classes or modules in a system coupling is an internal quality attribute which is to be calculated. Coupling is divided into three categories parameter coupling, global coupling and Inheritance coupling. There are no dynamic metrics available till date that can study the impact of inheritance on coupling. The study of any correlation existing between coupling and inheritance also needs to be studied. Coupling and inheritance are two key design attributes of object-oriented software systems that help predict external attributes of software quality.

Dynamic measures related to inheritance coupling are ambiguous for, e.g., type of coupling is an aspect in which cases which constitutes to a coupling are clearly not defined (Bidve and Khare, 2012). There is no clear picture of how to use inheritance in coupling. Every author has different idea regarding inherence for coupling measurement (Bidve and Khare, 2012).

2 Coupling metrics

To measure external complexity of a software design coupling is well recognised as one of the fundamental measure of quality. Coupling metric measure in turn characterises such external quality attributes of a design as its maintainability, reusability, reliability, and provides a way to estimate testing efforts (Bidve and Khare, 2012). The majority of coupling metric are evaluated through static code analysis but other dependencies regarding the dynamic behaviour of a program can only be inferred from run-time information. So, dynamic coupling metric came into picture. Features of object-oriented programming such as polymorphism, dynamic binding and inheritance render the static coupling metrics and were imprecise as they do not reflect perfectly the run-time situation.

Moreover, static coupling metrics attempt to predict the potential interactions that would take place at run-time, whereas dynamic coupling metrics measure what is actually happening at runtime rather than predicting.

Furthermore, with ever increasing use of object-oriented software in industry, it has been observed that inheritance and polymorphism are used more frequently to improve internal reuse in a system and facilitate maintenance.

The actual target of polymorphic method invocations can only be determined at run-time as per the inherited members of the class. Thus, it is not feasible to obtain precise static measures, which take inheritance and polymorphism into account (Gupta, 2011).

2.1 Coupling level

Coupling is defined as the degree of association between class through the use of methods or attributes defined in a class that are used by another class (Chawla and Nath, 2013). Classes interact with other classes to form a subsystem/system and this interaction can indicate the complexity of the design. Representative metrics at different coupling levels which are used are discussed below (Bidve and Khare, 2012):

2.1.1 Object level

Object level coupling between classes can be calculated how objects are associated and contributing in the complexity factor. Objects are instance of classes which helps classes to communicate with each other thus increasing dependency among them.

1 Coupling between objects (CBO) (Chidamber and Kemerer, 1994)

   CBO is indented to give an indication of the effort needed for maintenance and testing and defined as a count of the number of other classes to which it is coupled. Coupling between two classes is said to occur when one class uses methods or variables of another class. CBO considers the number of distinct non-inheritance related class hierarchies on which a class depends.

2.1.2 Class level

These metrics identify the highlighting aspects of the class abstractions and help identify where remedial action may be taken to remove abstraction. Representative metrics of this level are:

1 Response for a class (RFC) (Chidamber and Kemerer, 1994)

   According to C&K, RFC is a measure of the coupling of a class through the number of methods and the amount of communication with other classes as RFC is an indicator of the effort of testing and debugging. Methods that can be invoked in response to a message to an object of the class or by some method in the class are counted to give RFC measure. All methods accessible within the class hierarchy are included

2 Weighted methods per class (WMC) (Chidamber and Kemerer, 1994)

   If all methods are considered equally complex, then WMC is simply the number of methods defined in each class. WMC measures the complexity of an individual class.

C&K suggests that WMC is intended to measure how complex is the class with count of number of methods and it is an indicator of the effort needed to develop and maintain a class.

3 Inheritance metrics

Inheritance relationship is viewed as a trade-off between classes. Inheritance enhances association between the two class thus can be considered as a major contributor to calculate the coupling between the classes. Three aspects need to be considered with respect to inheritance (Chawla and Nath, 2013):

Is there a need to distinguish between inheritance-based coupling and non-inheritance-based coupling?

How do we assign methods and attributes to classes?

For method invocations: shall we consider static or polymorphic invocations.

The representative metrics of this level are:

1 Depth of inheritance tree (DIT) (Chidamber and Kemerer, 1994)

DIT measures the maximum level of the inheritance hierarchy of a class; the root of the inheritance tree inherits from no class and direct count of the levels of the levels in an inheritance hierarchy. According to C&K, DIT is intended to measure the class complexity, design complexity and the potential reuse because the deeper is a class, the greater number of methods is likely to inherit.

2 Number of children (NOC) (Chidamber and Kemerer, 1994)

NOC is the number of immediate subclasses to a class in the hierarchy and thus counts the number of subclasses belonging to a class. It is an indicator of the potential influence a class can have on the design of the system. If the design has a high dependence on reuse through inheritance, and may be better to split off functionality into several classes.

3 Number of inherited methods (Abreu and Melo, 1996)

NIM is a measure showing the reusable behaviour of a class. It counts the number of methods which a class can access in its super classes. Scope of this metric is a class. The larger the number of inherited methods is, the larger class reuse happens through sub classing. Thus it shows how much internal reuse happens between a class and its super classes based on invocations.

4 Method inheritance factor (MIF) (Abreu and Melo, 1996)

MIF is defined as the ratio of the sum of the inherited methods in all classes of the system under consideration to the total number of available methods for all classes. MIF measure directly the number of inherited methods as a proportion of the total number of methods. Abreu proposes that MIF is a measure of inheritance, and consequently as a means of measure the level of reuse thus gives assessment of testing needed.

5 Attribute inheritance factor (AIF) (Abreu and Melo, 1996)

AIF is defined as the ratio of the sum of inherited attributes in all classes of the system under consideration to the total number of available attributes. It is defined in an analogous fashion to MIF. AIF measure directly the number of inherited attributes as a proportion of the total number of attributes. Too much reuse trough inheritance makes worst the understandabilty and testability.

6 Polymorphism factor (PF) (Abreu and Melo, 1996)

PF is defined as the ratio of the actual number of possible different polymorphic situation for class Ci to the maximum number of possible distinct polymorphic situations for class Ci. Thus, giving a measure of polymorphism potential. PF is the number of methods that redefine inherited methods, divided by the maximum number of possible distinct polymorphic situations. Polymorphism arises from inheritance. Overriding methods can reduce complexity, thus increasing maintenance. It is an indirect measure of the dynamic binding in a system.

4 Complexity metric

Halstead complexity metrics quantitative measured the complexity directly from the operators and operands in the module to measure a program module's complexity directly from source code. Among the earliest software metric, they are strong indicators of code complexity. Because they are applied to code, they are most often used as maintenance metric. Because maintainability should be a concern during development, the Halstead measures should be considered for use during code development to follow complexity trends they are one of the oldest measures of program complexity (Suri and Garg, 2009).

Halstead metrics is based on interpreting the source code as a sequence of tokens and classifying each token to be an operator or an operand, counting:

1 Number of unique (distinct) operators (n1) number of unique (distinct) operands (n2)

2 Total number of operators (N1) total number of operands (N2).

3 The number of unique operators and operands (n1 and n2) as well as the total number of operators and operands (N1 and N2) are calculated by collecting the frequencies of each operator and operand token of the source program.

   a Program length (N)

   The program length (N) is the sum of the total number of operators and operands in the program:

   $$N = N1 + N2$$

   b Vocabulary size (n)

   The vocabulary size (n) is the sum of the number of unique operators and operands

   $$n = n1 + n2$$

c Volume (V)

- Program volume(V)

  The program volume (V) is the information contents of the program, measured in mathematical bits. It is calculated as the program length times the two-base logarithm of the vocabulary size (n):

  $$V = N * \log 2(n)$$

- Halstead's volume (V)

  Describes the size of the implementation of an algorithm. The computation of V is based on the number of operations performed and operands handled in the algorithm. Therefore, V is less sensitive to code layout than the lines-of-code measures.

d Difficulty level (D)

The difficulty level or error proneness (D) of the program is proportional to the number of unique operators in the program. D is also proportional to the ration between the total number of operands and the number of unique operands (i.e., if the same operands are used many times in the program, it is more prone to errors).

$$D = (n1 / 2) * (N2 / n2)$$

5 Proposed work

The proposed metric formula gives the coupling measure of a system due to methods that are inherited at runtime and the degree of association between them. With the help of this coupling value the measure of reuse can estimated as a low coupling value will give a more useable code, and thus by verifying values you can improve upon the code.

5.1 Proposed metric

5.1.1 Dynamic method inheritance coupling metric

$$DC_{micf}(c_i, c_j) = \text{Method inheritance coupling factor} * \text{coupling factor}$$

$$DC_{micf}(c_i, c_j) = \sum_{K=1}^{S} r_{mi}{}^{R}(c_i, c_j) * cf$$

where

$$\sum r_{mi}{}^{R}(c_i, c_j) = ovm + orefm / T$$

$$cf = (c_i, c_j = 1) / C^2 - C,$$

- cf is coupling factor
- ovm is the total number of overrided method and orefm is the total number of methods called through object reference

- S is the total number of methods invoked.
- T is the total number of methods
- (c_i, c_j = val) if c_i is associated with c_j, C is the no. of classes.

Relation $r_{mi}\,^{R}(c_i, c_j)$ is the number of inherited methods invoked and implemented in classes ci and cj through parameter called and overriding contributing to coupling.

s is the total number of invocation relations taking place between these classes at run-time.

The formula will give the total method inheritance coupling by calculating the method inheritance coupling factor which is calculated as summation of inherited methods/total methods. The inherited methods are calculated by two ways firstly by calculating the number of overridden methods contributing in inheritance coupling and secondly by calculating the method called by the reference objects contributing in parameter coupling. The coupling due to inheritance takes into consideration the number of interfaces implemented and classes extended each incrementing the measure by 1 for, e.g., class A extends B implements C, D, E. Coupling calculated would be 4.

5.1.2 Class reusability metric

Reusability metric will give the measure of the reusability in the Java class. Reusability is a software quality attribute calculated to improve software quality of the software system. Reuse is achieved by establishing various relationships among classes such as inheritance or composition. Features of class such as high generality level (i.e., less application specificity) and less coupling increases its reusability. Introducing reusability features in the classes must be the goal of inheritance hierarchy at design time.

Our approach is to derive formula to measure reusability of a class is based on following principles:

Inheritance is a direct indicator of reuse. The number of methods inherited (overridden, object reference) in a class will indicate the reusability of class due to inheritance. It indicates the measure of reusability in a class.

Applying least square regression analysis to calculate a and k

$$x = (\text{ovm} + \text{Orefm})$$

$$z = a + k * x$$

$$\text{Reusability metric} = z / T$$

$$a = z - k * x$$

Using empirical constant k the raw metrics values are empirically adjusted for calculation in order to map the metrics values into a scale from 0.0 to 1.00 (0 ± 100%).

X	addition of ovm is the total number of overrided method and orefm is the total number of methods called through object reference
T	is the total number of methods.

5.1.3 Inheritance coupling

A more independent a class it is easier to be reused by another application. To improve modularity and encapsulation the inter object class coupling measures should be minimum. As the number of couples becomes larger the maintenance is more difficult. By using more interfaces compared to inheritance the coupling measures are reduced.

Also, whenever one object interacts with another object, that is a coupling. Strong coupling means that one object is strongly coupled with the implementation details of another object. Strong coupling is discouraged because it results in less flexible, less scalable application software.

Thus, for my the proposed inheritance coupling measure in the tool ,the count of the coupling due to inheritance is calculated by assigning values on the basis of type of inheritance occurring between classes, i.e., value 1 has been assigned to the class with 'implementing interfaces' as it will contribute to least coupling, value 2 has been assigned to class using 'extend' keyword and value 3 has been assigned to the classes coupled through object calling which will contribute to highest coupling. And if class extends and implements a number the coupling value will result into 3. Thus, a low coupling measure will indicate that class can be reused and degree of inheritance is low.

Table 1 coupling measure

Type of inheritance	*Coupling value assigned*	*Coupling range*	*Coupling type*
Interface	1	1–10	Low
Extends	2	11–50	Moderate
Object calling	3	> 50	High

5.1.4 Class independent factor

The inheritance coupling for a Java project will give the measure of the amount of coupling , which exists in a class due to inheritance.

And as coupling and independency are inversely related as a class with high coupling measure will be less independent thus will be not easy to reuse. We have calculated an IF using the inheritance coupling measure given by the formula:

$$\text{Independent factor (class)} = 1 / \text{Inheritance coupling}$$

This factor will measure how much the class is independent.

Table 2 Independent factor measures

Value	*Independent factor*
FI	Fully independent
< 0.5	Less independent
> 0.5	Highly independent

5.1.5 Complexity

Tool contains metric factor option which contains complexity and coupling as the factors, any of them can be chosen and calculated for a chosen Java file. The complexity is measured based in the Halstead volume concept Halstead's measure derived from the unique and total number of operands and operators. In this tool program length measure of the Halstead metric has been considered and complexity is calculated based on the number of operands (identifiers and constants) and operators (traditional and keywords)

> 'if', 'else', 'while', 'case', 'for', 'switch', 'do',
> 'continue', 'break', '&&', '||', '?', ':', 'catch',
> 'finally', 'throw', 'throws', 'default', 'return',
> 'break;'

Any of the above mentioned operators and operands present in the code will be calculated to give a measure of complexity of the code. The complexity measure in the tool will basically gives the count of number of loops, few keywords that are basically used and making the code complex. By identifying a value the code with high value that will be identified as a highly complex code can be revised and thus efficiency can be improved.

Table 3 Measuring complexity

Complexity level	*Risk*
1–10	A simple program, without much risk
11–20	More complex, moderate risk
21–50	Highly complex, moderate risk
> 50	Most complex, very high risk

6 Implementation

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus, it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective. In this, we develop a tool in Java which explains as follows:

Metric Tool is a software tool developed in Java which takes Java code as input and measures dynamic method inheritance coupling metric, class reusability metric, inheritance coupling and complexity. In this tool, we create three steps in menu bar as shown in Figure 1 the steps are: file, metric factor, apply formula (Figure 1).

The working simulation result of the following Metric Tool is in details shown step by step.

1 The file menu in the tool can open a file, create a new file and save a file and help you exit the tool.

2 The Metric Factor menu in the tool will help you choose any of the two factors, i.e., complexity and coupling. Selecting any of them you can calculate a value for both separately for the chosen Java file.

3 The Apply Formula menu will calculate the metric value of the dynamic inheritance method metric and the reusability metric and independent factor.

Figure 1 Metric Tool (see online version for colours)

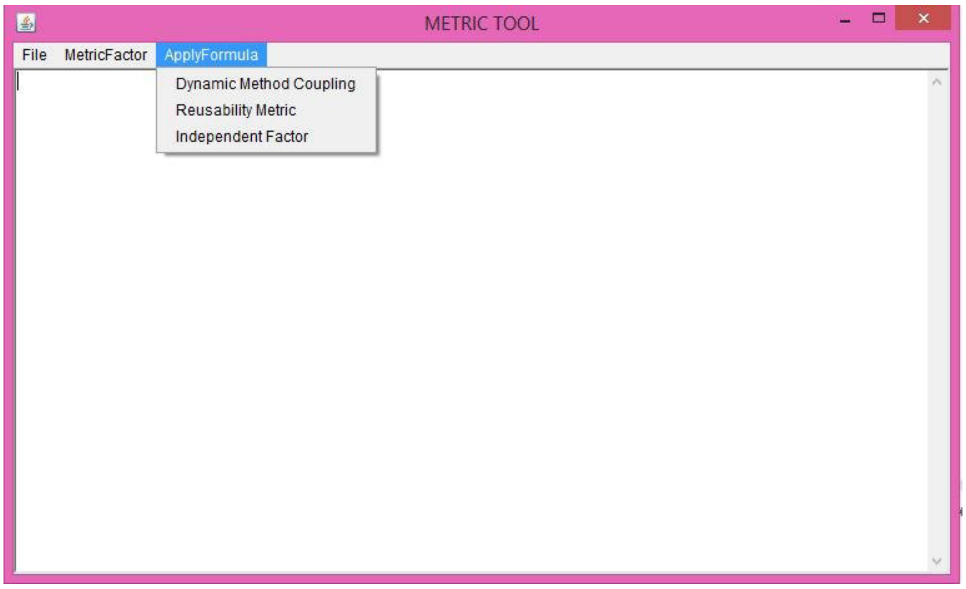
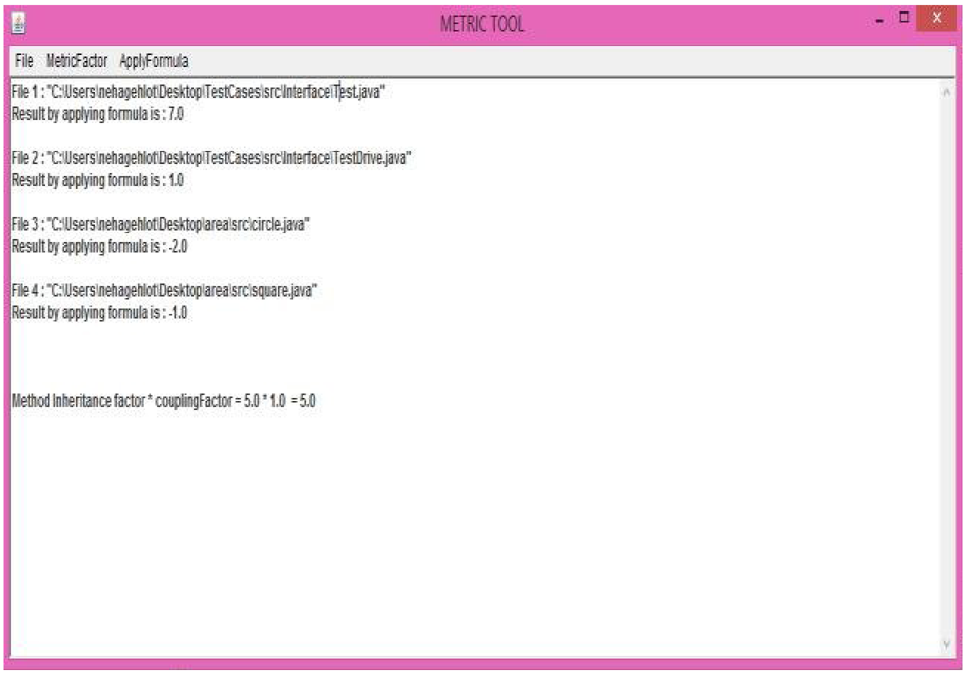
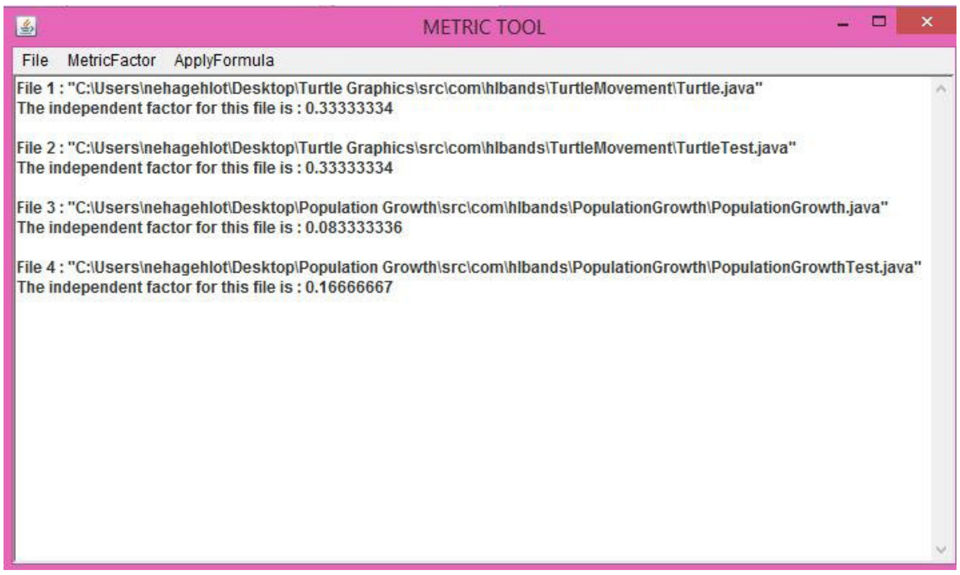


Figure 2 Tool calculating proposed metric (see online version for colours)



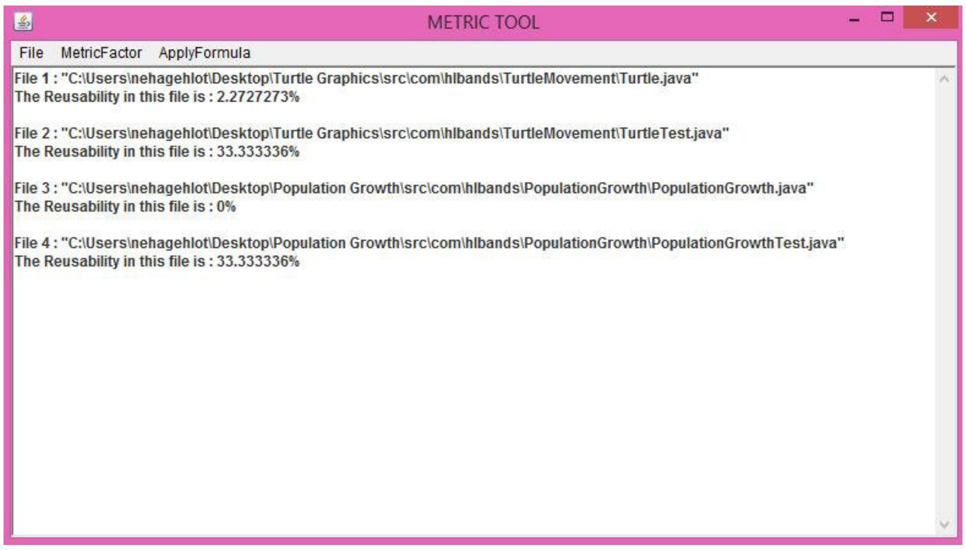
```
File 1 : "C:\Users\nehagehlot\Desktop\TestCases\src\Interface\Test.java"
Result by applying formula is : 7.0

File 2 : "C:\Users\nehagehlot\Desktop\TestCases\src\Interface\TestDrive.java"
Result by applying formula is : 1.0

File 3 : "C:\Users\nehagehlot\Desktop\area\src\circle.java"
Result by applying formula is : -2.0

File 4 : "C:\Users\nehagehlot\Desktop\area\src\square.java"
Result by applying formula is : -1.0



Method Inheritance factor * couplingFactor = 5.0 * 1.0  = 5.0
```

Figure 3 Tool measuring independent factor (see online version for colours)



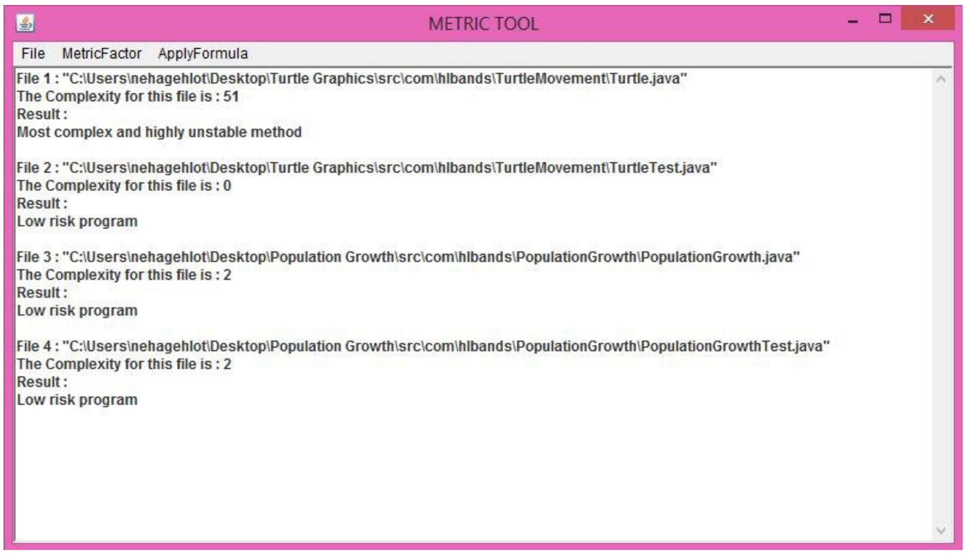
```
File 1 : "C:\Users\nehagehlot\Desktop\Turtle Graphics\src\com\hlbands\TurtleMovement\Turtle.java"
The independent factor for this file is : 0.33333334

File 2 : "C:\Users\nehagehlot\Desktop\Turtle Graphics\src\com\hlbands\TurtleMovement\TurtleTest.java"
The independent factor for this file is : 0.33333334

File 3 : "C:\Users\nehagehlot\Desktop\Population Growth\src\com\hlbands\PopulationGrowth\PopulationGrowth.java"
The independent factor for this file is : 0.083333336

File 4 : "C:\Users\nehagehlot\Desktop\Population Growth\src\com\hlbands\PopulationGrowth\PopulationGrowthTest.java"
The independent factor for this file is : 0.16666667
```

Figure 4 Tool measuring reusability metric (see online version for colours)



```
File 1 : "C:\Users\nehagehlot\Desktop\Turtle Graphics\src\com\hlbands\TurtleMovement\Turtle.java"
The Reusability in this file is : 2.2727273%

File 2 : "C:\Users\nehagehlot\Desktop\Turtle Graphics\src\com\hlbands\TurtleMovement\TurtleTest.java"
The Reusability in this file is : 33.333336%

File 3 : "C:\Users\nehagehlot\Desktop\Population Growth\src\com\hlbands\PopulationGrowth\PopulationGrowth.java"
The Reusability in this file is : 0%

File 4 : "C:\Users\nehagehlot\Desktop\Population Growth\src\com\hlbands\PopulationGrowth\PopulationGrowthTest.java"
The Reusability in this file is : 33.333336%
```

Figure 5 Tool measuring complexity (see online version for colours)



```
File 1 : "C:\Users\nehagehlot\Desktop\Turtle Graphics\src\com\hlbands\TurtleMovement\Turtle.java"
The Complexity for this file is : 51
Result :
Most complex and highly unstable method

File 2 : "C:\Users\nehagehlot\Desktop\Turtle Graphics\src\com\hlbands\TurtleMovement\TurtleTest.java"
The Complexity for this file is : 0
Result :
Low risk program

File 3 : "C:\Users\nehagehlot\Desktop\Population Growth\src\com\hlbands\PopulationGrowth\PopulationGrowth.java"
The Complexity for this file is : 2
Result :
Low risk program

File 4 : "C:\Users\nehagehlot\Desktop\Population Growth\src\com\hlbands\PopulationGrowth\PopulationGrowthTest.java"
The Complexity for this file is : 2
Result :
Low risk program
```

7 Result and analysis

7.1 Test cases

A number of Java projects has been used as test cases for evaluating the proposed metric, i.e., dynamic inheritance method coupling and metric tool which calculates complexity, inheritance coupling, and reusability metric. Total 30 Java projects with varying LOCs have been used, i.e., from simple Java projects to complex Java projects. Test cases chosen are standard Java codes with LOC from 6 to 5,000 has been tested. Test cases are mentioned with their classes in Table 3.

7.2 Test case analysis

Measure of reusability metric is also calculated and is related to amount of inheritance measure. The analysis shows measure of reusability in the code.

Measure of the proposed metric that is dynamic inheritance coupling metric is calculated for several test cases and presented. It is observed as the code in project increases the value of the metric increases to very high values due to presence of high degree of inheritance and high coupling within the code.

Table 1 shows the range values for the coupling due to inheritance defined by the tool.

Table 2 shows the range of class independent factor.

Table 3 shows the complexity range values defined in the tool.

7.3 Results

The complexity measure calculated here is based on Halstead volume concept and has taken program length as the factor to calculate the complexity due to presence of several operators and keywords. Results are shown in Table 4.

The reusability metric is defined as an inheritance measure and the results would show that the measure of existing reusability in a class and the class independent factor is calculated based on inheritance coupling measured in a class thus the factor calculated will give a measure that how much a class is independent allowing developers to select appropriate classes for reuse.

Results on several test cases are shown in Table 5. The dynamic inheritance method coupling metric calculates the coupling measure of inherited methods taking into consideration the overridden methods as well, which are not taken into earlier such proposed metrics and coupling factor which also includes coupling due to inheritance into its calculation. The coupling factor proposed earlier did not consider associations due to inheritance. Thus, code with less metric value is more reusable. Results of metric on several test cases are shown in Table 5.

Table 4 Table measuring complexity and coupling

Project	*Class*	*Complexity*	*Inheritance coupling*
String justify app	Stringjustify.java	24	0
	Stringjusttest.java	1	6
Show quality	Average.java	24	3
	Averagetest.java	1	3
Turtle graphics	Turtle.java	52	3
	Turtletest.java	1	3
Pythagorean table	Pythagorean.java	3	0
	Pythagore-antest.java	1	6
Population growth	Popula-tiongrowth.java	3	12
	Popula-tiongrwothtest.java	3	6
Knight tour	Knightour.java	12	6
	Knightourtest.java	1	6
Tic Tac Toe	Tictactoe.java	12	6
	Tictactoe test.java	1	6
Code	Signin.java	10	5
	Signup.java	7	5
	Testrun.java	2	0
	Uploadfile.java	7	5
	Uploadnew-file.java	9	5
App	Beeper.java	2	27
Online examination	Sample.java	12	30

Table 5 Table measuring metrics

Project	*Class*	*Independent factor*	*Reusability metric*	*Proposed metric*
String justify app	String justify.java	FI	0	2
	Stringjust-test.java	0.16	33.3	
Show quality	Average.java	0.33	5.5	1
	Averagetest.java	0.33	25	
Turtle graphics	Turtle.java	0.33	2.2	2
	Turtletest.java	0.33	33.3	
Pythagorean table	Pythagorean.java	FI	0	1
	Pythagoreantest.java	0.16	25	
Population growth	Populationgrowth.java	0.08	0	6
	Populationgrwothtest.java	0.16	33.33	
Knight tour toe	Knightour.java	0.16	4.5	8
	Knightourtest.java	0.16	37.5	
Tic Tac Toe	Tictactoe.java	0.16	4.1	4
	Tictactoetest.java	0.33	33.3	
Code	Signin.java	0.20	0	0.8
	Signup.java	0.20	0	
	Testrun.java	0.20	0	
	Uploadfile.java	0.20	30	
App	Uploadnewfile.java	0.20	30	84
	Beeper.java	0.03	42.2	
Online examination	Sample.java	0.33	48.4	infi

8 Conclusions

This paper gives a new dynamic method inheritance coupling metric that is implemented on a Metric Tool developed in Java. Also, the metric tool developed measures complexity which is based on Halstead volume concept, inheritance coupling measure, class independent factor and a reusability metric.

The complexity measure calculated here is based on Halstead volume concept and has taken program length as the factor to calculate the complexity. Thus, the code with less complexity will be comparatively easy to understand and thus used.

The reusability metric is defined as an inheritance measure and the results would show that the measure of existing reusability in a class. Due to the calculation of reusability metric the stability of the class structure can be calculated in future. The class independent factor will allow developers to choose appropriate classes for reuse.

The dynamic inheritance method coupling metric calculates the coupling measure due to inherited methods, also overridden methods, which are not taken into earlier such proposed metrics and coupling factor which also includes coupling due to inheritance into its calculation. Thus, code with less metric value is more reusable.

Coupling can also be used so that it enhances communication between the objects and thus increasing reuse while preserve in the scalability and flexibility. Inheritance used in the project is less and therefore it is more reusable. It helps in predicting that the inheritance used is not making the project complex and is reliable as well, thus reducing testing effort.

Acknowledgements

I would like to thank my guide Ms. Jagdeep Kaur and my H.O.D. Dr. Latika Singh for their understanding, patience and provided me with unending encouragement and support. Their mentorship was paramount in providing a well rounded experience for my long-term career goals. They encouraged me to not only grow as a Researcher also as an independent thinker. I would like to thank the Department of Computer Science at ITM University, especially for their input, valuable discussions and accessibility.

References

Abdellatief, M., Bakar, A., Sultan, M., Ghani1, A.A.A. and Jabar, M.A. (2012) ‘A mapping study to investigate component-based software system metrics’, *The Journal of Systems and Software*, 13 October, Vol. 86, No. 86, pp.587–603.

Abreu, e B.F. and Melo, W. (1996) ‘Evaluating the impact of object-oriented design on software quality’, *Proceedings of 3rd International Software Metrics Symp.*, Berlin.

Arisholm, E., Briand, L.C. and Foyen, A. (2004) ‘Dynamic coupling measurement for object-oriented software dynamic software’, *IEEE Transaction on Software Engineering*, Vol. 30, No. 8, pp.491–506.

Bidve, V.S. and Khare, A. (2012) ‘A survey of coupling measurement in object oriented system’, *International Journal of Advances in Engineering & Technology, ©IJAET*, January, Vol. 2, No. 1, pp.43–50, ISSN: 2231-1963.

Bidve, V.S. and Khare, A. (2012) ‘Simplified coupling metrics for object-oriented software’, *International Journal of Computer Science and Information Technologies (IJCSIT)*, Vol. 3, No. 2, pp.3839–3842.

Briand, L.C., Daly, J.W. and Wüst, J.K. (1999) ‘A unified framework for coupling measurement in object-oriented systems’, *IEEE Transactions on Software Engineering*, January/February, Vol. 25, No. 1.

Chawla, S. and Nath, R. (2013) ‘Evaluating inheritance and coupling metrics’, *International Journal of Engineering Trends and Technology (IJETT)*, July, Vol. 4, No. 7, p.2903.

Chidamber, S.R. and Kemerer, C.F. (1994) ‘A metric suite for object oriented design’, *IEEE Trans. Software Engineering*, Vol. 20, pp.476–493.

Chhabra, J.K. and Gupta, V. (2010) ‘A survey of dynamic software metrics’, *Journal of Computer Science and Technology*, Received June 12, 2008; Revised April 3, 2010, September, Vol. 25, No. 5, pp.1016–10259.

Chhikara, A. and Chhillar, R.S. (2012) ‘Analyzing the complexity of Java programs using object-oriented software metrics’, *JCSI International Journal of Computer Science*, January, Vol. 9, Nos. 1–3, pp.364–372.

Etzkorna, L.H., Hughes, W.E. Jr. and Davisa, C.G. (2000) ‘Automated reusability quality analysis of OO legacy software’, *Information and Software Technology*, April 2001, Vol. 43, No. 5, pp.295–308

Goulão, M. and Abreu, F.B. (2010) ‘Software components evaluation: an overview’, *International Journal of Computer Applications*, December, Vol. 40, No. 1, pp.1080–1196.

Gupta, D. (2013) 'Coupling based structural metrics – an quality assessment of software modularization', *International Journal of Advanced Research in Computer Science and Software Engineering*, July, Vol. 3, No. 6, pp.1796–1800.

Gupta, V. (2011) 'Validation of dynamic coupling metrics for object-oriented software', *ACM SIGSOFT Software Engineering Notes*, September, Vol. 36, No. 5, pp.2020976–2020985.

Hassoun, Y., Counsell, S. and Johnson, R. (2005) 'Dynamic coupling metric: proof of concept', *IEE Proc. – Softw.*, December, Vol. 152, No. 6, pp.273–279.

Honglei, T., Wei, S. and Yanan, Z. (2009) 'The research on software metrics and software complexity metrics', *International Forum on Computer Science – Technology and Application (IFCSTA'09),* IEEE Press, January, pp.131–136, doi:10.1109/IFCSTA.2009.39.

Rodriguez, D. and Harrison, R. (2001) 'An overview of object-oriented design metrics', *Proc. of the Conference on Software Technology and Engineering Practice (STEP),* IEEE Press, July, pp.230–237, ISBN 08186 78402.

Sharma, A., Kumar, R. and Grover, P.S. (2007) 'A critical survey of reusability aspects for component-based systems', *World Academy of Science, Engineering and Technology*, Vol. 33, pp.35–39.

Singh, P. and Singh, H. (2010) 'Class-level dynamic coupling metrics for static and dynamic analysis object-oriented systems', *International Journal of Information and Telecommunication Technology, IJITT*, July, Vol. 1, No. 1, ISSN: 0976-5972.

Souley, B. and Bata, B. (2013) 'A class coupling analyzer for Java programs', *West African Journal of Industrial and Academic Research*, June, Vol. 7, No. 1, p.13.

Suri, P.K. and Garg, N. (2009) 'Software reuse metrics: measuring component independence and its applicability in software reuse', *IJCSNS International Journal of Computer Science and Network Security*, May, Vol. 9, No. 5, pp.237–248.