5.1 Introduction

5.2 Categorization of Reuse and Reusability Metrics
   5.2.1 Related Work and Motivation
   5.2.2 Exploring Reuse and Reusability
   5.2.3 Inheritance Hierarchy Based Metrics
      5.2.3.1 Taxonomy of Inheritance Based Metrics
      5.2.3.2 Existing Metrics Selected for Comparison
      5.2.3.3 Proposed Metrics
      5.2.3.4 Case Study and Experimental Results

5.3 Correlating Dimensions of Inheritance Hierarchy with Complexity and Reuse
   5.3.1 Metrics and Hypotheses
   5.3.2 Experimental Results

5.4 Analysis of Experimental Results

5.5 Summary

5.1 INTRODUCTION
Software reuse is the use of existing artifacts to create new software and software reusability is the degree to which the artifacts can be reused. Software reuse reduces the development efforts and increases the quality of software. It is well recognized by both researchers and software developers that OOSD approach is capable of developing software by reusing existing classes and as well as for developing reusable classes. OOSD reuse the existing classes to fulfill the current requirements of customer as well as promise to develop the reusable classes to fulfill the future or changing requirements of customer. OOSD supports reuse in three ways (1) Verbatim reuse through instantiation and use of previously defined classes (2) Generic reuse through generic templates (3) Leveraged
reuse through inheritance [Karunanithi93]. Among these, inheritance is the foremost technique of reuse. Inheritance is the relationship among classes, wherein an object in a class acquires characteristics from one or more other classes [Roshanberg98]. By organizing classes into “classification hierarchy”, inheritance gives an extra dimension to the encapsulation of the abstract data types because it enables classes to inherit attributes and methods from other classes [Kadhim07]. Different inheritance hierarchies may be designed to implement inheritance for a problem therefore, it becomes essential to assess inheritance hierarchy from reuse and reusability point of views for increasing its potential benefits. In this regard this chapter performs following two tasks:

- Categorization of reuse and reusability metrics
- Correlating dimensions of inheritance hierarchy with complexity and reuse.

The first task is presented in section 5.2 and second task is presented in section 5.3.

5.2 CATEGORIZATION OF REUSE AND REUSABILITY METRICS

Many object oriented metrics are available in literature for assessing inheritance hierarchy but these metrics do not distinguish reuse and reusability. Reuse and reusability metrics evaluates two different aspects of inheritance hierarchy. Reuse metrics answers the question “What is the amount of reuse among classes in the software?” and reusability metrics answers the question “Whether the classes are reusable in future?”. This section investigates four existing metrics - DIT, NOC, MIF and AIF, and also proposes five new metrics - Breadth of Inheritance Tree (BIT), Method Reuse Per Inheritance Relation (MRPIR), Attribute Reuse Per Inheritance Relation (ARPIR), Generality of Class (GC) and Reuse Probability (RP). Proposed metrics are also compared with existing metrics. All inheritance based metrics are classified into two categories: Reuse Based Metrics (RBM) and Reusability Prediction Metrics (RPM). RBM are further classified into two categories - Reuse Indicator Metrics (RIM) and Reuse Estimation Metrics (REM).

5.2.1 Related Work and Motivation

Since the proposal of six object oriented metrics by Chidamber and Kemerer’s [CK94] many metrics have been proposed by various researchers. Reuse and reusability metrics
are focused by many studies. Abreu et al. [Abreu94b] defines many metrics among them two metrics - MIF and AIF measures the reuse using inherited methods and attributes. Sheldon et al. [Sheldon02] proposes metrics for understandability and modifiability of a class inheritance hierarchy and compare the proposed metrics with Chidamber & Kemerer’s and Handerson-Sellers’s metrics. Bhatia et al. [Bhatia08] define reusability of a class as a function of DIT, NOC and CBO. Authors state that reusability of whole class diagram is equal to the reusability of class having maximum reusability. DIT and NOC have positive effect on reusability whereas CBO has negative effect on reusability of a class. Gandhi et al. [Gandhi10] proposes four metrics (Number of Template Children (NTC), Depth of Template Tree (DTT), Method Template Inheritance Factor (MTIF) and Attribute Template Inheritance Factor (ATIF)) based on template and state that proposed metrics are reusability metrics. Sandhu et al. [Sandhu06] defines reusability in terms of tuned version of CK-metrics suite and proposes a neuro-fuzzy based model for automatic identification of reusability of object oriented software components. Rajnish et al. [Rajnish10] proposes three metrics namely - Derive Base Ratio Metric (DBRM), Average Number of Direct Child (ANDC) and Average Number of Indirect Child (ANIC) for measuring class inheritance tree. Authors compare proposed metrics with metrics proposed by Sheldon et al. and Henderson-Seller. Suri et al. [Suri09] defines reusability of a component in terms of its independency. Higher independency is treated as indicator of more reusability.

Most of the work in this field is related to proposal of inheritance based metrics and models. For better understanding of these metrics proper classification of metrics is required. This triggers the approach used in this study to classify the inheritance based metrics and to propose five new metrics.

5.2.2 Exploring Reuse and Reusability

Software reuse is the key activity in software development for improving the software productivity and quality by using existing software artifacts or knowledge to develop the new software. Effective software reuse can be only achieved through systematic quantified process. Effective reuse pays maximum productivity and quality benefits at fewer cost. Reusability is the property of any artifact that makes it reusable. Reusability
of any artifact can be defined as the degree to which it can be reused [Frakes96]. Morisio et al. defines software reuse as a systematic practice of developing software from a stock of building blocks [Morisio02]. Building blocks for reuse in OOSD are classes. OOSD includes both reuses of the existing classes as well as design of new classes that can be reused in future. 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. Concept of reuse and reusability can be understood from following example:

Suppose a web based software development company wants to analyze the optimization of its websites due to code, content and user interface. Code optimization deals with programming techniques to load and index the pages quickly. Effective contents of website deal with the keywords which can derive the visitors to the website. Good user interface makes the website interactive and user friendly. It is decided to design three classes: CodeOptimizationAnalyzer, ContentOptimizationAnalyzer and UserInterfaceOptimizationAnalyzer to measure the performance of websites due to code, content and user interface respectively. It is found that some of the features of these three classes are same therefore, by following the basic principle of designing of inheritance hierarchy that common features should be contained in super class, one more class OptimizationAnalyzer is designed which contains that common features. CodeOptimizationAnalyzer, ContentOptimizationAnalyzer and UserInterfaceOptimizationAnalyzer classes are derived from OptimizationAnalyzer class as shown in Figure 5.1.

OptimizationAnalyzer class is reused by CodeOptimizationAnalyzer, ContentOptimizationAnalyzer and UserInterfaceOptimizationAnalyzer classes. This inheritance hierarchy is reasonable as per the current requirements. However, in future the company may have to analyze the optimization from more different point of views such as graphics optimization, search engine optimization, database optimization etc. Therefore, the inheritance hierarchy should be more generalized so that more classes can be inherited in future. Another alternative of same problem is shown in Figure 5.2. Three
new classes: PerformanceOptimizationAnalyzer, BusinessOptimizationAnalyzer and DesignOptimizationAnalyzer are introduced to increase the reusability of inheritance hierarchy. Newly added classes in alternative-2 generalize the optimization analyzers and contain the common features of all the optimization analyzers which fall into same category. Alternative-2 provides more opportunity for reuse in future.

FIGURE 5.1 INHERITANCE HIERARCHY FOR WEBSITE OPTIMIZATION PROBLEM - ALTERNATIVE1

FIGURE 5.2 INHERITANCE HIERARCHY FOR WEBSITE OPTIMIZATION PROBLEM - ALTERNATIVE2
5.2.3 Inheritance Hierarchy Based Metrics

Designing inheritance hierarchy must be supported by reuse and reusability metrics to measure its effectiveness. Metrics can be used as indicator of improvement of reuse and reusability in inheritance hierarchy. Reuse metrics are used to compute the amount of reuse among classes and reusability metrics are used to predict the extent to which classes can be reused.

5.2.3.1 Taxonomy of Inheritance Based Metrics

For better understanding of inheritance based metrics it is necessary to classify them. This study categorizes inheritance based metrics into two categories - Reuse Based Metrics (RBM) and Reusability Prediction Metrics (RPM) as shown in Figure 5.3. RBM is further classified into two categories Reuse Indicator Metrics (RIM) and Reuse Estimation Metrics (REM). RIM of a class just gives the idea of reuse amount whereas REM actually computes the amount of reuse on the basis of method/attribute reused. RPM of a class computes the extent up to which it can be reused in future. RPM computes reusability only on the basis dimension and characteristics of inheritance tree. RPM don’t take account of actual number of method and attribute inherited in the computation.

Inheritance Based Metrics (IBM)

Reuse Based Metrics (RBM)  Reusability Prediction Metrics (RPM)

Reuse Indicator Metrics (RIM)  Reuse Estimation Metrics (REM)

FIGURE 5.3 TAXONOMY OF INHERITANCE BASED METRICS

Inheritance based metrics investigated and proposed in this chapter are listed in Table 5.1.
### TABLE 5.1 REUSE AND REUSABILITY METRICS

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Acronym</th>
<th>Source</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth of Inheritance Tree</td>
<td>DIT</td>
<td>CK Metrics [CK94]</td>
<td>RIM+RPM</td>
</tr>
<tr>
<td>Number of Children</td>
<td>NOC</td>
<td>CK Metrics [CK94]</td>
<td>RIM</td>
</tr>
<tr>
<td>Method Inheritance Factor</td>
<td>MIF</td>
<td>MOOD Metrics [Abreu94b] [Aggarwal06]</td>
<td>REM</td>
</tr>
<tr>
<td>Attribute Inheritance Factor</td>
<td>AIF</td>
<td>MOOD Metrics [Abreu94b] [Aggarwal06]</td>
<td>REM</td>
</tr>
<tr>
<td>Breadth of Inheritance Tree</td>
<td>BIT</td>
<td>Proposed</td>
<td>RIM</td>
</tr>
<tr>
<td>Method Reuse Per Inheritance Relation</td>
<td>MRPIR</td>
<td>Proposed</td>
<td>REM</td>
</tr>
<tr>
<td>Attribute Reuse Per Inheritance Relation</td>
<td>ARPIR</td>
<td>Proposed</td>
<td>REM</td>
</tr>
<tr>
<td>Generality of Class</td>
<td>GC</td>
<td>Proposed</td>
<td>RPM</td>
</tr>
<tr>
<td>Reuse Probability</td>
<td>RP</td>
<td>Proposed</td>
<td>RPM</td>
</tr>
</tbody>
</table>

#### 5.2.3.2 Existing Metrics Selected for Comparison

Many inheritance metrics are available in literature out of which four metrics which have been widely validated in literature are selected for further investigation and comparison with proposed metrics. Selected metrics are described as follows:

**5.2.3.2.1 Depth of Inheritance Tree (DIT)**

DIT of a class computes the maximum path length from that class to the root of the inheritance tree. Higher the DIT of a class, greater the number of methods it is likely to inherit, i.e. higher reuse. The class having higher DIT is less reusable as compared to class having lower DIT due to class at higher depth is more specialized as compared to class at lower depth in inheritance hierarchy. More specialized class provides less opportunity to inherit it. Therefore, DIT is categorized into RIM as well as RPM category.
5.2.3.2.2 Number of Children (NOC)

NOC of a class is the number of immediate sub-classes of that class. Greater number of subclasses/children is indicator of greater reuse. NOC indicates the reuse among classes therefore, it is RIM metric.

5.2.3.2.3 Method Inheritance Factor (MIF)

MIF computes the system level reuse in terms of total number of methods inherited and declared in the classes. Higher the value of MIF more is the amount of reuse in the software. Zero value of MIF indicates that there is no method inheritance in the classes, this may be due to the scope of methods don't permits method inheritance. MIF measures the amount of reuse in the inheritance hierarchy therefore, it is REM.

5.2.3.2.4 Attribute Inheritance Factor (AIF)

Like MIF, AIF is also system level metric and computes reuse in terms of total number of attributes inherited and declared in the classes. AIF is also categorized in REM category.

5.2.3.3 Proposed Metrics

Following are five proposed metrics.

5.2.3.3.1 Breadth of Inheritance Tree (BIT)

Breadth of inheritance tree is equal to the total number of leaf nodes in the inheritance hierarchy

\[
\text{BIT} = \text{Number of Leaf Nodes}
\]

For example BIT of inheritance hierarchy given in Figure 5.4 is 6 because it has six leaf nodes (E, I, J, C, G, H)

![Figure 5.4 Inheritance Hierarchy for Computing BIT](image-url)
BIT is indicator of reuse. Higher BIT means higher number of methods/attributes reused in the derived class. It don’t computes actual amount of inheritance but only indicates the reuse therefore, it is classified as RIM metric.

As compared to NOC metric which computes the number of immediate sub classes of a class, BIT measures the breadth of whole inheritance tree. As compared to DIT it gives another dimension to inheritance tree i.e. breadth. As compared to MIF and AIF, it doesn’t computes the actual number of method and attribute inherited therefore, categorized into RIM category.

5.2.3.3.2 Method Reuse Per Inheritance Relation (MRPIR)

MRPIR computes the total number of methods reused per inheritance relation in the inheritance hierarchy. It applies on whole inheritance hierarchy in the system. It can be computed as follows:

$$\text{MRPIR} = \frac{\sum_{k=1}^{r} MI_k}{r}$$

Where $r$ = Total number of inheritance relationships.

$MI_k$ = Number of methods inherited through $k$th inheritance relationship.

If same method is inherited through different inheritance relationships then it is computed separately in each relationship.

FIGURE 5.5 WEIGHTED INHERITANCE HIERARCHY FOR COMPUTING MRPIR AND ARPIR
Consider an inheritance hierarchy given in Figure 5.5. Weight of each arc is the number of methods and attributes inherited from base class to derived class. First number of weight represents number of methods inherited and second number of weight represents number of attributes inherited. MRPIR of this inheritance hierarchy is \( \frac{2+2+2+4+4+5+5+7+7}{9} = 4.22 \). This metrics is classified in REM category because it computes the amount of reuse by taking account of number of methods inherited in the inheritance hierarchy.

DIT and NOC are two dimensions of inheritance hierarchy and are useful indicator of reuse. However, MRPIR actually computes average number of method reused in the inheritance hierarchy. It gives clearer picture of reuse due to inheritance. As compared to MIF which considers method declared and inherited in all classes, MRPIR considers only reused methods and inheritance relationships only.

**5.2.3.3 Attribute Reuse Per Inheritance Relation (ARPIR)**

It computes the total number of attributes reused per inheritance relation in the inheritance hierarchy. Like MRPIR it is also categorized into REM category.

\[
ARPIR = \frac{\sum_{k=1}^{r} AI_k}{r}
\]

Where \( AI_k \) = Number of attributes inherited through kth inheritance relationship.

For example ARPIR of inheritance hierarchy given in Figure 5.5 is \( \frac{1+1+2+2+1+1+2+2}{9} = 1.44 \).

ARPIR is similar to MRPIR and can be compared with DIT, NOC and AIF in same way as MRPIR is compared with DIT, NOC and MIF.

**5.2.3.4 Generality of Class (GC)**

Generality of a class is the measure of its relative abstraction level. Higher the generality of a class more it is likely to be reused. GC can be computed as follows:

\[
GC = \frac{a}{al}
\]

Where \( a \) = Abstraction level of the class.

\( al \) = Total number of possible abstraction levels.

Higher the value of GC of a class means higher reusability. For example if
OptimizationAnalyzer class has three abstraction level (OptimizationAnalyzer, DesignOptimizationAnalyzer and UserInterfaceOptimizationAnalyzer) and the abstraction level of OptimizationAnalyzer, DesignOptimizationAnalyzer and UserInterfaceOptimizationAnalyzer is 3, 2 and 1 respectively, then their GC is 3/3, 2/3 and 1/3 respectively. OptimizationAnalyzer class is having the highest value of GC therefore, it is more reusable as compared to other two classes.

DIT metric considers only depth of a class in inheritance tree and takes higher depth as indicator of higher reuse. DIT does not consider characteristics of the class whereas GC considers the generality of the class as a feature of reusability. However, the relationship may exist between GC and DIT metric. A class with a small DIT has much potential for reuse as it tends to be a general abstract class [Chandra10]. As classes at higher depth are more specific as compared to the classes at lower depth therefore, higher depth indicates less abstraction level.

5.2.3.3.5 Reuse Probability (RP)

It is the probability of reusing classes in the inheritance hierarchy. It can be computed as follows:

\[
RP = \frac{N_i - N_{lg}}{N}
\]

Where

\(N_i\) = Total number of classes that can be inherited.

\(N_{lg}\) = Total number of classes that can be inherited but having lowest possible generic level.

\(N\) = Total number of classes in the inheritance hierarchy.

The final/sealed classes can't be inherited. The class having lowest generic level is most specialized class and assumed non inheritable. Higher the number of such classes in the software lower is the probability of reuse therefore, less reusability.

In best case RP=1, in this case all the classes are inheritable and their generic level allow them to inherit. In worst case RP=0, in this case all classes are non inheritable and having lowest generic level. Higher probability indicates more reusability of classes in inheritance hierarchy therefore, it is categorized into RPM category.

GC is depth and breadth independent metric. It computes reuse probability by taking
account of number of inheritable and non inheritable classes only. As it is RPM therefore, it does not require computation of method/attribute inherited and declared like in MIF and AIF metric.

5.2.3.4 Case Study and Experimental Results

One Case Study is designed to assess the metrics discussed in section 5.2.3.2 and 5.2.3.4. This Case Study is small and simple but shows the significance of metrics. The Case Study presents two alternative inheritance hierarchies of a software for educational institute as shown in Figure 5.6 and Figure 5.7. It is hypothesized that second alternative have more reuse and reusability as compared to first alternative. Generalization scale for Student class is shown in Figure 5.8. It is assumed that all methods are inheritable and attributes are non inheritable in both the alternatives. Results are presented in Table 5.2 and Table 5.3. In Table 5.2 class number 1, 2, 3, 4 and 5 represents Student, Bcastudent, Mcastudent, Bbastudent and Mbastudent respectively. In Table 5.3 class number 1, 2, 3, 4, 5, 6 and 7 represents Student, ITstudent, Mngstudent, Bcastudent, Mcastudent, Bbastudent and Mbastudent respectively.

![Inheritance Hierarchy Diagram](image)

**FIGURE 5.6 INHERITANCE HIERARCHY FOR EDUCATIONAL INSTITUTE - ALTERNATIVE 1**
FIGURE 5.7 INHERITANCE HIERARCHY FOR EDUCATIONAL INSTITUTE – ALTERNATIVE 2.

FIGURE 5.8 GENERALIZATION SCALE FOR STUDENT CLASS
### TABLE 5.2 RESULTS OF ALTERNATIVE-1

<table>
<thead>
<tr>
<th>Metric</th>
<th>Class No.</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>DIT</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NOC</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GC</td>
<td>1</td>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td>MIF</td>
<td></td>
<td></td>
<td>0.31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AIF</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRPIR</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARPIR</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP</td>
<td></td>
<td></td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### TABLE 5.3 RESULTS OF ALTERNATIVE-2

<table>
<thead>
<tr>
<th>Metric</th>
<th>Class No.</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>DIT</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>NOC</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GC</td>
<td>1</td>
<td>0.67</td>
<td>0.67</td>
<td>0.33</td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td>MIF</td>
<td></td>
<td></td>
<td></td>
<td>0.35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AIF</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRPIR</td>
<td></td>
<td></td>
<td></td>
<td>3.33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARPIR</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP</td>
<td></td>
<td></td>
<td></td>
<td>0.43</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.3 CORRELATING DIMENSIONS OF INHERITANCE HIERARCHY WITH COMPLEXITY AND REUSE

Inheritance hierarchy can be measured in two dimensions i.e. depth and breadth. Chidamber and Kemerer proposed metrics [CK94] - DIT and NOC for measuring depth and breadth of inheritance hierarchy respectively. Since the proposal of DIT and NOC, these have been widely validated by many researchers. It has been validated in literature that DIT and NOC are correlated with reuse, complexity, maintainability, fault-proneness and change proneness [Sheldon02][Gatrell10][Unger98][Daly96][Aggarwal07b][Briand00][Briand01][Briand06]. Authors also claim that deeper tree constitute higher complexity due to involvement of more methods and classes. Greater breadth is indicator of improper sub-classing and more testing efforts [CK94]. Higher reuse may affect complexity of software in terms of difficulty in understanding the classes. Higher reuse is always desirable whereas higher complexity is always undesirable for good quality software. DIT and NOC both measures the same feature of object oriented software but from two different perspectives. DIT and NOC do not measures the amount of reuse but only provides the depth and breadth of inheritance hierarchy respectively. Various other metrics such as MIF and AIF proposed by Abreu et al. [Abreu94b] are available which are based on actual amount of method and attribute reuse. The feedbacks of these metrics are very useful for improving the software design.

Daly et al. [Daly96] evaluates the effect of DIT on program maintenance and concludes that software having DIT equals to 3 are quicker to maintain as compared to software having no inheritance. Maintenance of software having DIT equals to 5 is slightly slower as compared to software having no inheritance. Unger et al. [Unger98] conducts the similar study with more complex and large software. Results suggest that higher DIT may complicate program understanding but DIT is not only factor for that, many other factors like complexity of program and type of maintenance tasks also affects maintenance time. Authors found high correlation between maintenance time and the number of methods trace to gain program understanding. Gatrell et al. [Gatrell10] conducts a empirical study using C# programs and concludes that class within specific range of DIT are more prone to changes and faults. Classes within specific range of NOC are more prone to changes.
while classes with a higher NOC are more fault-prone than those classes in a simpler inheritance hierarchy.

Following sections correlates the DIT and NOC with complexity and reuse. To measure method reuse and attribute reuse MIF and AIF metrics are used respectively. To measure complexity Response for a Class (RFC) is used.

### 5.3.1 Metrics and Hypotheses

Five design oriented metrics are used in this study. To measure the dimension of inheritance hierarchy DIT and NOC metrics are used. To measure the method reuse MIF metric is used and to measure the attribute reuse AIF metric is used. For measuring complexity of a class many metrics are available which measures the complexity from different perspectives such as size, relations, method complexity etc. Complex class is difficult to understand, test and maintain. Here RFC metric is used to compute the complexity of a class. RFC of a class is number of methods that can be potentially executed in response to a message received by an object of that class [CK94]. Higher RFC indicates testing and debugging of the class becomes more complicated since it requires a greater level of understanding on the part of the tester [CK94]. Therefore, higher RFC is analogous to higher complexity of the class. Originally RFC considers only methods of class and methods that can be directly invoked by methods of class. However, here we considers methods that can also be indirectly called by methods of class while computing RFC of a class which is denoted by RFC'. Different classes can have different RFC' in inheritance hierarchy therefore, maximum of RFC' is considered as RFC' of whole inheritance hierarchy.

Following four hypotheses (H1-H4) are considered:

**H1**: An inheritance hierarchy having more DIT than its peers is more complex as compared to them.

**H2**: An inheritance hierarchy having more DIT than its peers provides more method and attribute reuse as compared to them.

**H3**: An inheritance hierarchy having more NOC than its peers is more complex as compared to them.
H4: An Inheritance hierarchy having more NOC than its peers provides more method and attribute reuse as compared to them.

5.3.2 Experimental Results

Metrics are applied on two kinds of programs written in Java language. First category of programs includes programs having maximum NOC=1 but DIT varies from 0 to 6. Second category of programs includes programs having maximum DIT=1 but NOC varies from 0 to 6. For each program value of maximum RFC', MIF and AIF is computed. Results of metrics are presented in Table 5.4 to Table 5.7.

**TABLE 5.4 RESULTS OF METRICS AT NOC=1**

<table>
<thead>
<tr>
<th>Metric</th>
<th>DIT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Max(RFC')</td>
<td>2</td>
</tr>
<tr>
<td>MIF</td>
<td>0</td>
</tr>
<tr>
<td>AIF</td>
<td>0</td>
</tr>
</tbody>
</table>

**TABLE 5.5 RESULTS OF METRICS AT DIT=1**

<table>
<thead>
<tr>
<th>Metric</th>
<th>NOC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Max(RFC')</td>
<td>2</td>
</tr>
<tr>
<td>MIF</td>
<td>0</td>
</tr>
<tr>
<td>AIF</td>
<td>0</td>
</tr>
</tbody>
</table>
TABLE 5.6 CORRELATION OF DIT WITH REUSE AND COMPLEXITY METRICS

<table>
<thead>
<tr>
<th>Metric</th>
<th>Correlation Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC</td>
<td>0.97</td>
</tr>
<tr>
<td>MIF</td>
<td>0.55</td>
</tr>
<tr>
<td>AIF</td>
<td>0.30</td>
</tr>
</tbody>
</table>

TABLE 5.7 CORRELATION OF NOC WITH REUSE AND COMPLEXITY METRICS

<table>
<thead>
<tr>
<th>Metric</th>
<th>Correlation Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC</td>
<td>0.07</td>
</tr>
<tr>
<td>MIF</td>
<td>0.94</td>
</tr>
<tr>
<td>AIF</td>
<td>0.98</td>
</tr>
</tbody>
</table>

5.4 ANALYSIS OF EXPERIMENTAL RESULTS

Analysis of experimental results obtained from Section 5.2.3.4

Maximum DIT of alternative-2 (Figure 5.7) is higher than alternative-1 (Figure 5.6) therefore, it indicates higher reuse. BIT of both the alternatives are same however alternative-2 has more number of inherited methods as compared to alternative-1. Maximum NOC of alternative-1 is higher as compared to alternative-2 but overall total number of subclasses in alternative-1 is 4 which is less than total number of subclasses of alternative-2. MIF of alternative-2 is higher as compared to alternative-1. Therefore, higher amount of methods are reused in alternative-2. MRPIR of alternative-2 is higher than MRPIR of alternative-1. Therefore, alternative-2 has more reuse as compared to alternative-1. AIF and ARPIR of both alternatives are 0 because no attribute inheritance is considered. RP of alternative-2 is higher as compared to RP of alternative-1. Therefore, alternative-2 is more reusable as compared to alternative-1. Classes of alternative-1 have only two distinct values of GC (1, 0.33) whereas classes of alternative-2 have three different values of GC (1, 0.67, and 0.33). Average value of GC of alternative-2 (0.52) is higher than average value of GC of alternative-1 (0.46). Therefore, alternative-2 is more
Analysis of experimental results obtained from Section 5.3.2

- The value of Max(RFC') is increasing with the increase in DIT but value of Max(RFC') may increase or decrease with the increase of NOC. Therefore, increase in DIT makes the inheritance hierarchy complicated but it is not same for NOC.
- Increase in DIT increases the value of MIF. This indicates higher value of DIT means higher method reuse. Increase in NOC also increases the value of MIF which indicates higher NOC cause higher method reuse.
- Increase in DIT need not necessary increase the value of AIF. With the increase of DIT value of AIF may increase or decrease. But results show that higher value of NOC yields higher AIF.
- In most of the cases MIF is higher than AIF because generally methods are freer to access as compared to attributes. Attributes are generally restrictive to access.

**Correlation of DIT with Complexity**
Correlation coefficient (0.97) shows that DIT is strongly positively correlated with maximum RFC' i.e. higher DIT causes higher complexity. Therefore, hypothesis H1 is accepted.

**Correlation of DIT with Method and Attribute Reuse**
Correlation coefficient between DIT and MIF (0.55), correlation coefficient between DIT and AIF (0.22) shows that increase in DIT also increase method and attribute reuse. Therefore, hypothesis H2 is also accepted. However DIT is not strongly correlated with AIF.

**Correlation of NOC with Complexity**
Correlation coefficient (0.07) shows that NOC is very poorly correlated with RFC'. It shows that higher NOC does not mean higher complexity. Therefore, hypothesis H3 is rejected.

**Correlation of NOC with Method and Attribute Reuse**
Correlation coefficient between NOC and MIF (0.94), correlation coefficient between NOC and AIF (0.98) shows that NOC is strongly positively correlated method and
attribute reuse. Higher NOC means higher method and attribute reuse. Therefore, hypothesis H4 is accepted.

5.5 SUMMARY

Extensive use of OOP in software development is due to ability of developing more maintainable and reusable products. The major feature of OOSD is development with reuse. Large and complex software can be developed easily and quickly using OOP as compared to traditional procedure oriented paradigm. Inheritance is the major feature of object oriented approach to implement reuse. Most of the coding in object oriented software is involved in form of inheritance. At design level reuse is modeled by inheritance hierarchy. During designing of object oriented software lots of efforts are made to adjust inheritance hierarchy and relationships between classes to increase the reuse and reusability. Inheritance hierarchy is the major design artifact used for measuring the reuse and reusability in object oriented systems. It provides early opportunity of measuring reuse and reusability during software development.

This chapter evaluates inheritance hierarchy by existing as well as new metrics. Concept of reuse and reusability is differentiated and appropriate existing metrics are suggested for evaluating reuse and reusability. Five new metrics are also proposed and compared with existing metrics. Newly proposed metric are - Breadth of Inheritance Tree (BIT), Method Reuse Per Inheritance Relation (MRPIR), Attribute Reuse Per Inheritance Relation (ARPIR), Generality of Class (GC) and Reuse Probability (RP). New metrics are compared with four existing metrics - DIT, NOC, MIF and AIF. Taxonomy of inheritance hierarchy based metrics is also given for better understanding. Validity of reuse and reusability metrics is validated using a Case Study. Another task performed in this chapter includes establishing relationships among dimensions of inheritance hierarchy with reuse and complexity using five design oriented metrics - DIT, NOC, MIF, AIF and RFC. In this task four hypotheses (H1-H4) are tested, out of which H3 is rejected and all other hypotheses are accepted based on results obtained.