CHAPTER 4

OBJECT ORIENTED STRUCTURE
IMPROVEMENT METRICS

4.1 INTRODUCTION

The Quality Indicator is the pioneering process to evaluate the Software Quality of software. It was developed to help all software developers gain more value from the design of object-oriented software and to assist in improving the quality of design. A software quality indicator can be calculated to provide an indication of the quality of the system by assessing the system characteristics. In this chapter design metrics and Quality Indicator standards are proposed and developed, which assists a software development team to define and check the evolution of Software Quality at the key stages in the development process.

4.2 NEED FOR METRICS

The shortcomings of the existing metrics discussed in chapter 2 and the need for new metrics especially designed for object-oriented software have been suggested by a number of authors. The challenge is therefore to propose metrics that are firmly rooted in theory and which are relevant to practitioners in organizations. Hence, these needs are addressed through the development and implementation to verify a new suite of metrics for object-oriented design. Seven design metrics and two proposed metrics are analytically evaluated against the proposed Software Assessment Model. An automated Software Quality Indication was then developed and implemented to collect an empirical sample of these metrics to demonstrate their feasibility. These suggest the ways in which managers can use metrics to improve the
quality of design. Also, the impact of the proposed metrics in the Software Assessment Model is highlighted.

Two object-oriented design metrics are proposed, namely Effort Estimate metrics and Cost Estimate metrics for object-oriented systems. Theoretical basis for the development of object-oriented metrics is shown in Figure 4.1. When software complexity evaluation is performed before system building, it can be used for predicting the key aspects like development, testing and early maintenance costs. Moreover, on the basis of knowledge that is present in the early stages of the software life-cycle, for example the number of classes, main class relationships, number of methods, method interfaces, the process of system analysis and design allows for the defining and tuning of metrics for predicting effort. From the cognitive point of view, the observable complexity can be regarded as the effort required to understand the subsystem or class behaviour and functionalities. This complexity can usually be evaluated in the early phases of design and can be used for predicting the costs of reuse and maintenance (Cant & Jeffery 1994) for estimating and predicting other features (Nesi & Campanai 1996); (Nesi & Querci 1998); (Nesi 1998).

As a result, a model and metrics are proposed for the calculation of effort estimation or prediction of early design. The validation was performed by using real data from different projects. The validation presented shows that several metrics can be profitably employed for effort estimation or prediction can also be successfully used in the estimation or prediction of early Effort Estimate by using the model. The metrics presented in the work are an extension of a framework specifically defined for object-oriented systems (Fioravanti et al. 2001); (Fioravanti et al. 1998); (Ramaraj et al. 2007). The Effort Estimate metrics calculates the total size of the system S, by calculating the number of lines of code of each class C. Here component (C) implies the
classes, use cases and the events. The research focusses on Events, State and Use Cases and verifies their efforts. The RUP (Rational Unified Process) of UML is based on the use case concept.

The Cost Estimate metrics identifies the key class or the supporting class for the redesigning effort intimated by the Software Assessment Tool. The key class has close attachment to most of the elements in the project than the supporting class and redesigning a key class will involve significantly more effort. The Cost Estimate also considers the real time issues for the Cost Estimate, the project type, the developer skills and the user interface type, which are also found to have an influence over the efforts in the development.

**Figure 4.1 Theoretical Basis for the Development of Object Oriented Metrics**

4.2.1 **Effort Estimate Metrics (EE)**

Effort Estimate metrics measures the size of design elements, typically by counting the elements contained within, for example, the number of operations in a class, the number of classes in a package and so on. Size metrics are good candidates for developing the cost or Cost Estimates for implementation, review, testing or maintenance purposes and activities. Such
estimates are then used as input for project planning purposes and personnel allocation.

In addition, large design elements are big classes or packages may suffer from poor design. In an iterative development process, more functionality is added to a class or package over the time. The risk is that, eventually, many unrelated responsibilities are assigned to a design element. As a result, it has low functional cohesion. This in turn negatively impacts the understandability, reusability and maintainability of the design element. Therefore interfaces and implementation of large classes or packages should be reviewed for functional cohesion. If there is no justification for the large size, the design element should be considered for refactoring, for instance, extract parts of the functionality to separate, more cohesive classes.

Practical Software Measurement (PSM) (Card 2003) defines a generic process for selecting software measures and using the resulting information to manage projects. The research is based on a broad survey of the related literature that can assess the state of the art and practice in object-oriented measurement along with modelling and also maps the information collected in the PSM framework. It proposes a metrics called the Effort Estimate Metrics which uses the events, states and Use cases, as those elements in the project decides the scope of the object-oriented programming under investigation.

\[
\text{Size (S)} = \text{Size (C)}
\]

where Components (S) = \{C_1 \ldots C_n\}, such that if C_i = C_j then i=j, where \( i, j = 1 \ldots n \). The above equation calculates the total size of the System S by calculating number of lines in the code of each class C.
Here, Components (S) implies the classes, use cases and the events. Various researches have been carried out on design metrics and researchers have developed Quality Model for Objected Oriented Design (QMOOD) (Bansiya & Davis 2002). It measures the functional, structural and relational details of the system based on high-level attributes. In the proposed model, the reusability is calculated based on the coupling, cohesion, Effort Estimate and messaging.

The Proposed Effort Estimate Metrics

The proposed work takes into consideration, the Effort Estimate metrics, comfortable for most of the OOPS project that are successfully proved in the real time, based on the quality control suggestions from the Company and has opted this metrics for testing. The Effort Estimate metrics can be calculated as follows

\[
\text{Effort Estimate} = \text{No. of classes} \times \text{No. of methods} \times \text{No. of variables.}
\]

The resultant value for Effort Estimate should not exceed the standard, which is based on the experience study from the industry. The tool analyses the input sample classes for various metrics (Seven Metrics) and Effort Estimate metrics, which has the major influence for the design quality indications. The Software Assessment tool will intimate Java classes, which has a deviation from the standard and suggests for re Effort Estimates for those classes.

The research focusses on Events, States and Use cases and has verified the efforts. The RUP (Rational Unified Process) part of UML is based on the use case concept. A use case captures a contract between the stakeholders, also called the primary actors of a system and its behaviour. The use case gathers the different sequences of behaviour, or scenario together. In
short, an use case is a good way of eliciting requirements at the early stages of software development.

**Number of Actors associated with a Use case (NAU)**

This metric computes the number of actors which are associated with a use case and it is useful to measure the importance of the requirement expressed by the use case. The reason for this argument is that, the requirements that many actors concerned are likely to be important for the system to function properly as a whole. It does not count normal system classes for this metric, because this metric concerns the interactions between systems and its stakeholders.

**Number of Messages associated with a Use case (NMU)**

As explained before, a use case is further refined through its scenario. In UML, there are two scenario diagrams, which are the sequence diagram and the collaboration diagram. These two kinds of scenario diagram are completely isomorphic in meaning, where, one kind of diagram can be automatically replaced with another kind without any loss of information contained in it. The NMU metric counts the number of messages comprising the scenario of a use case. This metric is useful for tracing requirements into design-level elements.

**Number of System Classes associated with a Use case (NSCU)**

This metric counts the number of classes whose objects participate in the scenario of a use case. This metric does not include actors as this is done with the NAU metric. Like NMU, NSCU metrics are good for estimating the impact of a requirement change onto the system. Any change in the use cases, spread to classes, the interactions of their objects and vice
versa. The Effort Estimate is calculated based on all the above parameters events, states and use case (NAU, NMU, NSCU) of the classes involved.

4.2.2 The Cost Estimate Metrics (CE)

Software project managers are responsible for controlling the project budgets and are responsible to estimate the effort required for software development in its total life cycle. The principal components of project costs are the hardware costs, travel cost, training costs and effort costs. The dominant cost is the effort cost, which is most difficult to estimate and control. It has the most significant effect on the overall costs.

Many software systems built in the recent years have been developed using the object-oriented technology. In most cases, the estimation and prediction of effort is performed with difficulty due to the lack of metrics and suitable models.

The work also covers OOPS software effort estimation which predicts the redesign or rework efforts incurred as a result of design imperfections. The estimation typically produces projections of the software size, effort, schedule and the quality required to complete the project. These estimates form the basis for initial project plans and subsequent re-plans. Generally, the same factors that get estimated need to be tracked in order to manage a successful project.

While estimation models may provide from 15 to 100 adjustment factors, most organizations realize that their performance is only affected by a few of them. The survey of estimation models for object-oriented development shows that the two most commonly used adjustment factors are the team experience or expertise and the application type are complexity or
difficulty. If complexity is considered as an adjustment during the software sizing step, then it should not be factored again.

Comparison of results from (Nesi & Querci 1998) and (Lorenz & Kidd 1994) also suggests that the effort to implement a class in Smalltalk is greater than the effort to implement a class in Java. Bucci et al. (1998) notes that the productivity declines as teams get larger than four persons. Documentation still needs to be considered in planning object-oriented projects. Nesi & Querci (1998) report that 36% of effort is spent on documentation for an industrial project.

The Proposed Cost Estimate Metrics

Estimating object-oriented projects has a great demand, even though people have been estimating object-oriented projects are implemented in many different programming languages for over a decade. The following are some of the things that have been identified. The simple estimates are based on the number and kind of classes contained in the object-oriented application.

The experience and skill of the object-oriented developer is the second most important factor in this estimate. According to the industrial estimate, a skilled developer codes an OOPS project class in 10 days and a beginner will do it in 20 days. This difference in duration confirms that the developer’s skill and effort is a significant factor towards the completion of the project. The project type is also an important factor, which ranges to any size. The number of developers and their effort is given as an input to the project type. Design wise, the proposed Cost Estimate identifies the key class or the supporting class in the redesigning effort intimated by the Software Assessment Tool.
The key class has close attachment to most of the elements in the project than the supporting class and redesigning a key class involves significantly more efforts. The Cost Estimate also take into consideration the real time issues like the project type, developer skills and the user interface type which also are found to have an influence over the efforts in the development.

\[ \text{Cost Estimate} = K \times U \times Ex \]

Number of key classes (K), Complexity created by the user interface (U) and Experience of the developer (Ex).

As suggested earlier during redesigning, the efforts depend on the subjects taken for redesigning. Also when the User Interface type involves drag and drop facilities, the system involves more complexity from real time exceptions. The experience of the developer is based on the time response of the developer to arrive at a Effort Estimate for a class.

The Project manager carries a statistics of the developer profile. An industry procedure is taken and an attempt is made on a real time consideration regarding the cost per hour while calculating the redesigning efforts.

4.2.3 The Cost Estimate Calculation

In calculating the Cost Estimate for the OOPS project using the Design Phase Model, the input source is verified for the OOPS Metrics for any design compliance and imperfections. The model automatically indicates the classes that require redesigning efforts based on the practical standard. In addition, it accepts certain details from the user. The model considers the key
classes, supporting classes, the anticipated complexity of user interface as well as the developer efficiency to arrive at the Cost Estimate.

The estimation of effort is a complex task since it considers various parameters for its calculation. The four important parameters such as User Interface type, Project size, Expert skill set and Class type are taken as important parameters along with the per effort cost. Each of the parameters stated varies and takes values ranging from minimum to maximum is depicted in Figure 4.2.

**Figure 4.2 Cost Estimate Calculations**

The first key parameter, which helps in cost estimation, is the User Interface. The design of the user interface by the programmer can be simple or complex depending on the nature of the project. Hence, the Cost Estimate metrics can take the minimum or maximum value of the parameter for an application. The effort or cost estimate also depends upon the size of the projects. When the size of the project is small, it assumes a minimum value and in the case of others, it assumes either the medium or maximum value.
The next parameter taken for consideration is the expert skill set which according to many companies is difficult to obtain. The code efficiency of a project depends on the knowledge of the programmer in developing solution to it. Hence, the expertise or skill set required for a programmer plays an important role in cost or effort estimation. Depending on the programmer’s expertise, the project may take values between low, medium or high and thereby affects the estimation of effort.

The last parameter which helps in effort estimation is the class type. In designing the classes, the classes can be categorized as key classes or support classes. The type of classes indicated also forms a major factor in effort estimation as it indicates the attachment of classes to the project or application. All the parameters are proposed to be equally important. These parameters are implemented for the effort estimation based on the per effort cost and produce the results shown in Figure 4.3.

Figure 4.3 Cost Estimate calculating the redesigning efforts for a project
4.2.4 The Ranking Approach

A Ranking based mostly performance is planned to perform annotation prediction that is taken into account as user’s potential annotation for bugs. The method takes the information area unit obtained from Existing package modules with noted defect numbers as well as values of all package metrics which has lines of code, previous defects, and new lines side, and also the numbers of defects found. Once finishing ranking approach on defect modules, results area unit created in standout sheet.

4.2.5 System Analysis and Cost Estimation

Cost Estimate supports deciding is needed for the allocation of funds to program areas and/or specific comes details like techniques are accustomed estimate effort, metrics are accustomed live the accuracy of effort estimation methods/techniques. What effort predictors are utilized in computer code development method.

4.3 SOFTWARE QUALITY INDICATOR

Software maintenance is inevitable if software systems need to remain useful in their environment. Changes are necessary to continuously increase the value of software. Major portion of software maintenance activities is devoted to the modification of the software. Due to the coupling between software artifacts, changes made to one artifact can ripple throughout the software, requiring further changes to other artifacts. The ISO/IEC 9126 software quality standard (ISO 1991) defines six quality attributes of software like functionality, reliability, efficiency, usability, portability and maintainability. According to this standard, maintainability has four quality attributes as analyzability, changeability, testability and stability. Stability is
defined as those attributes of software that bear on the risk of unexpected effect of modifications.

Some stability measures were proposed for procedural programs (Yau & Collofello 1985); (Yau & Chang 1984) and for object-oriented design (Samadzadeh & Khan 1994); (Boehm & Basili 2001). There are some weaknesses in the existing measures that prevent their wide acceptance. The object-oriented design metrics proposed (Chidamber & Kemerer 1994); (Chidamber & Kemerer 1991) were adopted as candidate indicators of the logical reliability of the object-oriented designs. These metrics were selected because they are well defined, widely used and are shown to be valid quality indicators. The objective of the indicator study is to investigate whether or not, there are correlations between these metrics and the logical reliability of classes.

4.3.1 The Related Quality Indicators

The posteriori metrics (Nesi & Querci 1998) are calculated from the implemented software projects, where a full range of detailed parameters can be derived from the fully implemented code and design. These metrics are useful for examining the quality of the system and relating its final properties to the earlier predictive models and estimations. The metrics can be used for the testing and evaluation of the system and contribute to the ongoing evolution and maintenance. Predictive metrics derived in the early pre-coding Software Assessment or during the course of implementation can be used both in project planning as effort estimators and early quality indicators (Basili et al. 1996); (Nesi and Campanai 1996). Early defect prediction is a useful goal, which allows identifying high risk components which are 'expensive' to implement or error prone (EI Emam et al. 2001).
4.3.2 The Proposed Quality Indicator

The Software Quality Indicator methodology suggests an optimal standard for the major metrics found during the Software Assessment. It validates and compares the given input design against the standard values. The standards were based on the practical feedback collected from several industrial object-oriented developers and followed by industries.

Table 4.1 Quality Indicator Standards

<table>
<thead>
<tr>
<th>METRICS</th>
<th>Quality Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBO</td>
<td>8</td>
</tr>
<tr>
<td>Weighted attributes / Class</td>
<td>5</td>
</tr>
<tr>
<td>Response for a class</td>
<td>35 instances</td>
</tr>
<tr>
<td>Lines of code</td>
<td>50 lines</td>
</tr>
<tr>
<td>LCOM</td>
<td>1</td>
</tr>
<tr>
<td>DIT</td>
<td>6 level deep</td>
</tr>
<tr>
<td>WMC</td>
<td>15 methods</td>
</tr>
<tr>
<td>Effort Estimate</td>
<td>45 Counts</td>
</tr>
<tr>
<td>Cost Estimate</td>
<td>1 class – 10 days - 15 days redesigning effort</td>
</tr>
</tbody>
</table>

Based on the values, it indicates the imperfections and provides suggestions for redesign when it deviates from the standard arrived by the Quality Indicator which is shown in Table 4.1.
Based on the quality of design, Quality Indicator suggests the following:

If the output of the Software Assessment Model

- Exceed the Quality Indicator standards, the system strongly recommend for redesign.
- Equal to Quality Indicator standards, the system needs still more correction in the design.
- When it is less than the Quality Indicator standards, the system strongly agrees to Quality Indicator and is considered as being stable, maintainable and easily understandable with reduced maintenance cost.

When the Quality Indicator value is not confirmed by the project, the value in comparison with the standard value is indicated for the development of the project. The stability and maintainability of the project is assessed, which accounts for the major cost prevention during maintenance due to imperfect designs. The model also takes into account all the redesign classes and calculates the redesign cost efforts based on the metrics suggested (Lorenz & Jeff 1994) for better optimization of the project design at an earlier development stage.

**Reliability of Object Oriented Design**

As changes are made to an object-oriented design, its structure or behaviour may be affected. Modifications made to one class can have ripple effects on other classes in the design. Ripple effects may or may not be desirable do not require additional changes. A good object-oriented design from stable standpoint should localize changes as much as possible to classes on which alterations are made. The reliability of an object-oriented design
indicates its resistance to interclass propagation of changes that the design would have when it is modified. Class reliability is the likelihood that the class will not be change-prone as a consequence of changes made to other classes in the design. There are two aspects of design reliability. Logical reliability, that is concerned with the stability of the design structure and performance reliability, which is concerned with the reliability of design behaviour. The role of object-oriented design metrics are discussed in the development of software by measuring the key parameters involved in the project and a high level Software quality indicator is designed to indicate the quality of design at its early stages.

4.3.3 Software Defect Prediction Using Quality Indicator

A software system metric could be a quantitative lives of a degree to that a computer code or method possesses some property. Since quantitative measurements area unit essential altogether sciences, there's an eternal effort by computing practitioners to bring similar approaches to software system development. The goal is getting objective, duplicate and quantitative measurements, which can have various valuable applications in schedule and budget coming up with, value estimation, quality assurance testing, software system debugging, software system performance optimization, and best personnel task assignments.

Software Defect Prediction (SDP) is developed as a learning downside in code engineering, which has drawn growing interest from each domain and trade. Static code attributes square measure extracted from previous releases of code with the log files of defects, and accustomed build models to predict defective modules for consecutive unleash. It helps to find elements of the code that square measure a lot of doubtless to contain defects. This effort is especially helpful once the project budget is restricted, or the
The entire package is simply too giant to be tested thoroughly. An honest defect predictor will guide code engineers to focus the testing on defect-prone elements of the code. Software defect prediction will also facilitate to assign testing resources expeditiously through ranking software system modules in keeping with their defects. The present software system defect prediction models that square measure optimized to predict expressly variety the quantity of defects associate degree in a terribly software system module may fail to provide the correct order as a result of it's very troublesome to predict the precise number of defects in an exceedingly software system module attributable to unclear information.

This method is all regarding predicting defects victimization object directed metrics and version history for every module. When the prediction method is over, the modules square measure being hierarchic in keeping with their severity and therefore the overall value for the trouble is calculable. Severity may be a parameter to denote the implication of defect on the system – however essential defect is and what's the impact of those defects on the full system’s practicality. The severity may be a parameter set by the tester whereas he opens a defect and is principally up to speed of the tester. Sample example for defect prediction is depicted in Table 4.2.

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Lines of Code</th>
<th>Previous Defects</th>
<th>Lines Added</th>
<th>Defect Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>456</td>
<td>4</td>
<td>45</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>123</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>156</td>
<td>1</td>
<td>23</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>321</td>
<td>3</td>
<td>23</td>
<td>Unknown</td>
</tr>
<tr>
<td>E</td>
<td>211</td>
<td>2</td>
<td>43</td>
<td>Unknown</td>
</tr>
<tr>
<td>F</td>
<td>2354</td>
<td>15</td>
<td>432</td>
<td>Unknown</td>
</tr>
</tbody>
</table>
4.4 SUMMARY

The ranking task helps to portion testing resources additional expeditiously by predicting that modules area unit seemingly to possess additional defects. SDP information area units are collected by completely different firms and by different folks, that area unit noisy. As a result, predicting an explicit range of defects for every code module is difficult or perhaps not possible thanks to the dearth of correct historical information. during this paper, have applied Out-Of- Bag(OOB) approach to classify the most effective attributes from the information set for the predictions. Additionally thought-about the thing familiarized (OO) metrics to live against the disposition. Within the future work, we'll apply additional metric choice strategies to analyse additional deeply the effectiveness of metrics for SDP for the ranking task.

Thus, the Effort Estimate and Cost Estimate metrics were proposed and implemented in the design level before the development and coding stage. The model calculates the Effort Estimate by using the number of classes, the number of methods and the number of variables. Also, the working of design metrics in the model was explained. The need for effort calculation in the design stage and working of the effort estimation by using the Cost Estimate metrics were discussed. The reliability of the model was also highlighted.