CLASS QUALITY EVALUATION USING CLASS QUALITY SCORECARDS

Dr. E. Chandra¹, Ms. P. Edith Linda²

¹(Director, Computer Applications, Dr. SNS Rajalakshmi College of Arts and Science, Coimbatore, crcspeech@gmail.com)
²(Assistant Professor, School of IT and Science, Dr G. R. Damodaran College of Science, Coimbatore, p.lindavinod@gmail.com)

ABSTRACT

A major problem faced by the software professionals is the inability to produce the correct reliable software within the specified duration and budget. These failures are caused by the complex nature of the problem description and due to improper insight of the deliverables which is to be produced after the specified task. These problems can be reduced to an extent with the introduction of software metrics. The problem mainly arises because of the failure to measure the common set of properties in the development process. Therefore the introduction of software metrics alone cannot solve the problem completely but also the utilization of the metrics set will provide a partial solution to the problem faced by the professionals in the software metrics. The purpose of this paper is to study and analyze the various metric suites for object oriented systems, and hence examine the existing parameters of the suite and their contribution to software quality. Then design and develop a software prototype called “Class Break point Analyzer (CBA)” for extracting the parameters of the studied metric suites. Then build “Class Quality Scorecards” to study the contribution of these parameters to software quality.

Keywords: Metrics, Quality, Measurement, LCOM, WMC, MHF, CBO

1. INTRODUCTION

Metrics are units of measurement. Metrics is used to measure the quality of software programs and plays a vital role while developing them. Each and every thing which is used needs measurement in order to know its quality and quantity. Quality and quantity are the two different parameters which is used to measure the programs or software. Quality is used to
measure how good the software is whereas quantity is used to measure the number of programs or product involved. Quality is the ongoing process of building and sustaining relationships by assessing, anticipating, and fulfilling stated and implied needs. The first definition of software metrics is proposed by Norman Fenton “Software metrics is a collective term used to describe the very wide range of activities concerned with measurement in software engineering. These activities range from producing numbers that characterize properties of software code through to models that help predict software resource requirement and software quality”[16].

Software metrics can be applied to every process of software development. This metrics can be used to analyze the design of the software and code. The main motivation of the metrics is to improve the process of developing software and identify potential defects through quantitative measures. The metrics are mainly used to show the quality related issues while programming the object oriented environment. Software metrics are an integral part of the state-of-the-practice in software engineering. Measurements are used extensively in areas of production and manufacturing to estimate costs, quality and performance factor. The figure below depicts the principle according to Fenton [9], “Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules”. There are four important scopes of metrics which are correctness, maintainability, integrity and usability. Scope of these metrics are mainly focused to develop a software without any flaws and for the secured maintenance

**SCOPE OF METRICS**

**CORRECTNESS**

**MAINTAINABILITY**

**INTEGRITY**

**USABILITY**

**Fig.1 Scope of Metrics**

Correctness: The degree to which a program operates according to specification and maintainability of the object oriented program. Maintainability: The degree to which a program is open to change. Program that is easy to maintain, potentially saves large costs. A program can be maintained either informally or as a function of directly measured. Integrity: The degree to which a program is resistant to outside attack. There are many hackers who can attack the program. Though object oriented programs are more secured, some algorithms can attack that too. In those cases integrity has to be maintained. Usability: The degree of easiness to use. The users who are using the tool should be flexible enough to use with. The software or program should be easy to understand and easy to use[20].
The Goals of using Software Metrics are:

- Better understanding of product quality
- To assess process effectiveness
- To improve quality of the work performed at the project level.

2. SURVEY OF LITERATURE

Everything has to be measured in order to quantify it. Definite and valid measurements have to be specified to for the strength and weakness of the software metrics work [12]. When dealing with software metrics, the major point of concern is what to measure, how to measure and the contribution of the measurement to the quality of the software. Typical care should be taken only to collect the key metrics that aid in building the organization rather than the metrics that would demotivate the employees of the organization. Thus care should be taken to select the metrics that serve the organization towards better planning. Metrics should serve as the quantiative basis for making quality decisions [10]. Software metrics can be mainly categorized into traditional metrics and object oriented metrics. The metrics suite for Object Oriented design was proposed by Chidamber and Kemerer and it addressed the shortcomings of the traditional metrics[7]. The focus on process improvement and the demands for the software measures lead to the introduction of an object oriented measurement suite called CK metrics suite[7]. They proposed six CK design metrics including Weight Method Per Class (WMC), Number of Children (NOC), Depth of Inheritance Tree (DIT), Coupling Between Object class (CBO), Response For a Class (RFC), and Lack of Cohesion in Methods (LCOM). In a work done by Parvinder Singh Sandhu and Dr. Hardeep Singh [11] stated that the traditional metrics is not enough to the software developed in object oriented language and hence the necessity of the object oriented metrics suite was felt. When the CK metrics a mostly widely accepted metrics suite was tested, it could not satisfy certain axioms specified by called Weyukar’s axioms[18]. Their work was to propose extensions and refinements to the CK metrics suite for meeting the object orientation properties.

Basili et. al. [3][4] were among the first to validate these CK metrics using some C++ systems developed by students. They demonstrated the usefulness of CK metrics over code metrics. In 1998, Chidamber, Darcy and Kemerer explored the relationship between the CK metrics to productivity, rework effort or design effort separately [8].They show that CK metrics have better explanatory power than traditional code metrics based on three economic variables.

Raed Shatnawi [14] applied thresholds on CK metrics suit parameters to find the error-prone classes of the system. The threshold values for the parameters CBO, RFC and WMC were applied to open source projects like Eclipse version 3.0 at two levels of risk and the fault prone classes were identified. Univariate Binary Logistic Regression was used to examine whether there exists any association between the metric and error- proneness of the classes. Benlarbi et al. [5] estimated the threshold values of OO metrics using Ulm's statistical method. This study was intended to predict the effect of threshold on the fault proneness of the classes. The study indicated that the thresholds had no effect on the fault proneness of the classes. One inference out of the study stated that “as long the complexity of the classes is low, the classes are easily understandable”. The results indicate that no value for the studied CK measures where the fault-proneness changes from being steady to rapidly increasing.
Subramanyam and Krishnan investigated three design metrics, Weight Method Per Class (WMC), Coupling Between Object Class (CBO), and Depth of inheritance Tree (DIT), to predict software faults [15]. The system they study is a large B2C e-commerce application suite developed using C++ and Java. They showed that these design metrics are significantly related to defects and that defects are strongly related to the language used. Nagappan, Ball and Zeller in [9] predict component failures using OO metrics in five Microsoft software systems. Their results show that these metrics are suitable to predict software defects. Recovering design from source code has been demonstrated by G. Antoniol et al. in software reverse engineering [1]. Briand et.al. demonstrated recovering of sequence diagrams, conditions and data flow from Java code by using transformation techniques [6].

3. PROBLEM SPECIFICATION

Many applications which are developed goes off shelf when it reaches it saturation limit. But the reasons which lead to the saturation of the software have to be analyzed. The crucial reason is due to a number of changes made in the software due to policy changes or government policy. For example, fees collection procedure in an educational institution is not collected once a year from their students, but the government issues order to collect the fee annually from them. In this situation if a software system is used, then certainly some changes in the code have to be incorporated in to meet the new user requirement. But most of the occasions the changes are not reflected back in the design document. When changes are made repeatedly, it results in the saturation of the software. The use of these kinds of software contributes to erroneous results and this makes them go off-shelf and hence not utilizable. Many users and organization lost money and time due saturation of software and non operative software. Hence the objective of the work is to make the off shelf components reusable by making changes to the existing components using thresholds. To make the off shelf component usable, the component is broken into the corresponding classes which contribute the component. Metric based quality management processes usually define project-relative thresholds. They find relative outliers, i.e., system parts with metric values extremely large or small when related to the average values in the whole system. Moreover, they find negative tendencies over the development process, i.e., system parts with metric values increasing (decreasing) over time while small (high) values actually correlate with good quality. However, project-relative thresholds are critical unless the majority of system parts is in good quality. Decomposition of the component into the corresponding classes and analyzing it with thresholds gives conclusion as to which classes has to be changes to yield better quality and reusability.

Organizations has to fine tune the work of the junior programmers with experienced programmers for better quality. This tool aims at evaluating the work of the junior programmers using the threshold values applied to code developed by them. The benefit is that, some parts of the code which contribute to the poor quality can be restructured before combining the same to the software package under development. Here a tool called “Class Breakpoint Analyzer” is developed to solve both the problems as mentioned above.

In this research a proposed tool called “Class Break Point Analyzer” to evaluate the code quality of the software. Most of the works mentioned in the literature are covering the metrics that can be computed at the design level. This tool could be used in the maintenance phase or even the developers could use this tool to find the adherence of code to the object
oriented designing principles. The input to the tool would be the source code of existing software. Another type of input to the tool can be the code developed by the programmer. In the first case the tool can be used to evaluate the breakthrough point of the class and in the second case the tool can be used as a self-evaluation tool to the programmer. The tool uses the metric parameters as mentioned in CK and MOOD metric suite.

4. DESIGN AND DEVELOPMENT OF CLASS BREAKPOINT ANALYZER (CBA)

The focus and the purpose of this section is to discuss the design and the development of the proposed tool called the Class Breakpoint Analyzer. The main objective of this tool is to determine the breakpoint of a single class. A single class cannot contain all the attributes and functionalities. Hence the class has to be broken in several classes depending on the functionality for easy maintenance and reusability. Therefore the proposed tool gives suggestions on the parameters where the class is overloaded and gives the developers an indication that the class is saturated with lot of responsibility. The parameters which are determined are the same as mentioned in CK metrics suite and some of the parameters as mentioned in the MOOD metrics suite[11,13]. Threshold value of the parameters are the determination factor to suggest whether the class is overload or not and a Scorecard is generated for every class which determines whether the class has to be revamped or not. [2,19]

4.1 Parameters Extracted with the proposed Class Breakpoint Analyzer

4.1.1. Weighted Methods Per class (WMC)

It is a count of sum of complexities of all methods in a class. To calculate the complexity of a class, the specific complexity metric that is chosen (e.g., cyclomatic complexity) should be normalized so that nominal complexity for a method takes on value 1.0. Consider a class $K1$, with methods $M1, \ldots, M_n$ that are defined in the class. Let $C_1, \ldots, C_n$ be the complexity of the methods[7].

$$WMC = \sum C_i \ldots C_n \quad ----- \ (1)$$

For the sake of simplicity one can assume that the complexity of all the class is the same. Hence WMC is the sum of all the methods in the class. The threshold limit is set to 15 per class.

4.1.2. Depth of Inheritance Tree (DIT)

It assess how deep, a class is in hierarchy structure. This metric assesses the potential reuse of a class and its probable ease of maintenance. A class with a small DIT has much potential for reuse it tends to be a general abstract class. On the other side, as a class gets deeper into a class hierarchy, it becomes more difficult to maintain due to the increased mental burden needed to capture it functionally.

$$DIT = \text{maximum inheritance path from the class to the root class.} \quad ----(2)$$

This work suggests that lower DIT has great potential of reuse; hence a threshold value of 6 levels is set for DIT.
4.1.3. Number of Children (NOC)

It is a simple measure of the number of classes associated with a given class using an inheritance relationship. It could be used to assess the potential influence that a class has on the overall design. NOC measures how many classes inherit directly methods or fields from a super-class. Here in the system, the value of NOC to be 6 as same as DIT.

\[
\text{NOC} = \text{number of immediate sub-classes of a class} \quad -----(3)
\]

4.1.4. Lack of Cohesion in Methods (LCOM)

It is the difference between the number of methods whose similarity is zero and not zero. The similarity of two methods is the numbers of attributes used were common. LCOM can judge the cohesiveness among the class methods. Low LCOM indicates high cohesiveness and vice versa. High LCOM indicates that a class shall be considered for splitting into two or more classes. The LCOM takes its values in the range 0 to 1 [2]. The computation of LCOM is as follows:

\[
\text{LCOM} = 1 - \sum \frac{MF}{(M \times F)} \quad -----(4)
\]

Where,

- M is the number of methods in class (both static and instance methods are counted, it includes also constructors, properties getters/setters, events add/remove methods).
- F is the number of instance fields in the class.
- MF is the number of methods of the class accessing a particular instance field.
- Sum(MF) is the sum of MF over all instance fields of the class.

4.1.5. Coupling between objects (CBO)

When 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. OO metrics can help you to measure the right level of coupling. Coupling (MIC) is defined as the relative number of classes that receive messages from a particular class.

\[
\text{MIC} = \frac{nMIC}{(N - 1)} \quad -----(5)
\]

Where,

- N = total number of classes defined within the project.
- nMIC = total number of classes that receive a message from the target class

4.1.6. Response for Class (RFC)

It is defined as a count of the set of methods that can be potentially executed in response to a message received by an instance of the class.

\[
\text{RFC} = |RS| \quad -----(6)
\]

Where,

- RS is the response set for the class

4.1.7. The Attribute Hiding Factor (AHF)

It is the ratio of the sum of the invisibilities of all attributes defined in all classes to the total number of attributes defined in the system under The AHF is computed by dividing the number of invisible attributes by the number of all attributes [10].

\[
\text{AHF} = \frac{IA}{\sum AA} \quad -----(7)
\]

Where,
Both the measures MHF and AHF can provide overall system view about amount of information hiding incorporated by software designers [17].

4.1.8. The Method Hiding Factor (AHF)

It is the ratio of the sum of the invisibilities of all methods defined in all classes to the total number of attributes defined in the system under The MHF is computed by dividing the number of invisible attributes by the number of all attributes[10].

$$MHF = \frac{IM}{\sum AM} \quad (8)$$

Where,

- IM is the number of invisible methods
- AM is the total number of methods

4.1.9. Method Inheritance Factor (MIF)

It represents the percentage of effective inheritance of methods. The MIF is computed by dividing the number of all inherited methods in all classes by the sum of all methods available (inherited and locally defined) in all classes. MIF is computed using the formula

$$MIF = \frac{\sum IM}{\sum AM} \quad (16)$$

Where,

- IM is the inherited methods
- AM is all methods of the class

4.1.10. Attribute Inheritance Factor (AIF)

It represents the percentage of effective inheritance of attributes. The AIF is computed by dividing the number of all inherited attributes in all classes by the sum of all attributes available (inherited and locally defined) in all classes. AIF is computed using the formula

$$AIF = \frac{\sum IA}{\sum AA} \quad (17)$$

Where

- IA is the inherited attributes
- AA is all attributes of the class

5. DESIGN AND DEVELOPMENT OF CLASS QUALITY SCORE CARD

The proposed Class Quality Scorecard(CQS) measures the quality related parameters of the software in terms of Reusability, understandability, Functionality, Effectiveness and Extendibility. The Quality attributes are framed to measurable criteria which are in turn are the metric parameters which are extracted from the source code of the components or the Java programs. The table mentioned gives the desirable quality attributes of the software as mentioned by Bansiya and Davis in [2]. The Design properties are measures in terms of
Design Size, abstraction, Encapsulation, Coupling, Cohesion, Composition, Inheritance, Polymorphism, Messaging and Complexity [2].

Table 1 Quality Attribute Explanations

<table>
<thead>
<tr>
<th>S.No</th>
<th>Quality Attribute</th>
<th>Explanation</th>
<th>Necessary Metric Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Reusability</td>
<td>Reflects the presence of object-oriented design characteristics that allow a design to be reapplied to a new problem without significant effort.</td>
<td>NOC, DS, CBO, LCOM, RFC</td>
</tr>
<tr>
<td>2.</td>
<td>Understandability</td>
<td>The properties of the design that enable it to be easily learned and comprehended. This directly relates to the complexity of the design structure.</td>
<td>DS, NOC, WMC, MHF, AHF, LCOM</td>
</tr>
<tr>
<td>3.</td>
<td>Functionality</td>
<td>The responsibilities assigned to the classes of a design, which are made available by the classes through their public interfaces.</td>
<td>DS, NOC, AIF, MIF</td>
</tr>
<tr>
<td>4.</td>
<td>Effectiveness</td>
<td>This refers to a design’s ability to achieve the desired functionality and behavior using object-oriented design concepts and techniques.</td>
<td>DS, NOC, DIT, AIF, MIF, NOM, NOC</td>
</tr>
<tr>
<td>5.</td>
<td>Extendibility</td>
<td>Refers to the presence and usage of properties in an existing design that allow for the incorporation of new requirements in the design.</td>
<td>MHF, AHF, DIT, AIF, MIF</td>
</tr>
</tbody>
</table>

6. SAMPLE RESULTS

This section highlights some of the same results as generated by the Class Quality Scorecard. The Scorecard coloring procedure is as follows:

- If the output value is within the threshold value then the cell have to be in green color. The green color indicates the safe level.
- If the deviation value is between 25% then the cells have to be in yellow color i.e., warning stage.
- If the deviation is more than the 25% of the threshold, then the cell will be in red color, i.e., it shows the danger state. The following class quality scorecards are generated using the Class Breakpoint Analyzer for a Banking application.

Fig.3 Class Quality Scorecard generated using Class Breakpoint Analyzer for a class loan.java
The following charts are generated for the individual parameters. The following sample table and chart represent the value for NOM and DIT for the various classes in the file Bank.jar.

Table 2 Represents the values of the parameter NOM for the file Bank.jar

<table>
<thead>
<tr>
<th>ClassName</th>
<th>NOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.bank</td>
<td>1</td>
</tr>
<tr>
<td>com.customer</td>
<td>1</td>
</tr>
<tr>
<td>com.deposit</td>
<td>2</td>
</tr>
<tr>
<td>com.loan</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig.4 Represents the values of the parameter NOM for the file Bank.jar

Table 3 Represents the values of the parameter DIT for the file Bank.jar

<table>
<thead>
<tr>
<th>ClassName</th>
<th>DIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.bank</td>
<td>1</td>
</tr>
<tr>
<td>com.customer</td>
<td>1</td>
</tr>
<tr>
<td>com.deposit</td>
<td>2</td>
</tr>
<tr>
<td>com.loan</td>
<td>3</td>
</tr>
</tbody>
</table>

Fig.5 Represents the values of the parameter DIT for the file Bank.jar

7. CONCLUSION

1. The selected parameters are adequate enough to access the Java programs.
2. The developed system is capable of finding the weakness in each program and provides feedback to the developer to enhance the quality of the program.
3. The system defines the problem in each class exactly as the measurements are done class wise.
4. Thresholds on the parameter values are highly helpful for marking the quality of the classes.

REFERENCES

Journal Papers


Books:


Proceedings Papers: