Abstract

This paper provides a survey on quality assessment of the object oriented software. UML diagram plays an important role for the quality attribute assessment by extracting various metrics and gives an idea of the quality of the software. And to give a tabular representation to look on different key factors relating to quality assessment of object oriented software.

I. INTRODUCTION

The Quality of the software is the measure of its reliability and to calculate the quality of object oriented software there are two categories that is code-dependant and code-independent quality assessment. Code-dependant quality can be measured after the coding phase of the software development whereas Code-Independent quality is measured after the design phase of the software development. There are three main sections in which first section explains the role of UML diagram in quality assessment. Second section gives the relation between quality attributes and quality metrics of the Object oriented Software and finally the third section explains various models that have been proposed.

II. LITERATURE SURVEY

An In recent time there are many methods are proposed and many approaches have been outlined. And for the relevance in [2] authors have proposed the information about measuring the metrics of Object Oriented Software (OOS) development and proposed a Metric Measurement Model (MMM) to ensure that metrics of quality attributes are very important in Object Oriented Software improvement. The quality of OOS design...
can be enhanced by assessing the object oriented design metrics. The OO design was done by UML diagrams; particularly class diagram holds the important part of designing in the development process. They provided the information about various metrics of Object Orientation and measurement of those metrics using QMOOD[9] computations the main objective of the paper is to calculate the quality attributes that are mainly Effectiveness, Functionality, Extedibility, Reusability, Understandability And Flexibility using various object oriented design metrics such as Average number of classes (AC), Direct coupling (DC), Cohesion between methods (CBM), Total number of methods (TM), and number of public methods (NPM).

Another Runtime Complexity based on Method Access Points (RuCMAp) method was proposed in [3]. In the study, the runtime behaviour of their proposed metric on an open source software package Rhino 1.7R4 was examined. They computed and validated parameters with code clones and bug data with a very high complexity. They presented that code clone and bug data are worse for the quality analysis and to determine the software quality various software engineering complexity metrics are used and then compared with the pre-defined complexity threshold. And if determined complexity is more than predefined values then following result is determined with aspects like:

1. Low system quality
2. Failure
3. Defects
4. Cost
5. Maintenance

A formula was used to calculate the runtime complexity is as follows

$$RuCMAp = \frac{\sum_{i=1}^{F} F_i}{\sum_{j=1}^{O} O_j}$$  \hspace{1cm} (1)

Where F = Functions accessed at runtime, and O = Object instantiated.

In addition predicting fault proneness is also good to achieve high quality of software by change impact analysis given in [1] in which they proposed an extended the approach for SCIA[10] that integrates both change impact set and fault-proneness.

The use of a model to describe the behaviour of a system is a proven and major advantage to test. In [12], main focus is on model-based testing. The Model-based testing is equivalent to test case derivation from a model showing software behaviour. Test case design from requirement specification refers to Black box testing where as code based testing is white box testing, but Model based testing is termed as Grey Box testing. In this approach Test effort can be calculated or analyzed by Test Cases, Testing Execution, Testing Evaluation and Test Case Triplets[S I O] is used where:

- S = State
- I = Input
- O = Expected output

Model based Testing explains that there are many problems in code based testing to assess the behaviour of the software such as state of software cannot be evaluated. Where as in model based testing the extraction of state of the software is very easy.
UML diagrams are easily available for the design model. In one section they presented the various models used in software testing, that are as follows:

1. UML (Unified Modelling Language)
2. FSM (Finite State Machine)
3. Markov Chains
4. Grammars

As in [13] differentiation of software complexity metrics in accordance with the procedural and object oriented approach of programming languages. Software developers and maintainers need to read and understand source programs. The increase in size and complexity of software affects several quality attributes, especially understandability and maintainability. In this paper they discussed various procedural and object oriented software metrics that are as follows:

1. Process (during the process development)
2. Product (Output)
3. Resource (Input at the requirement phase)

And also tried to calculate complexity of sample code by using different procedural metrics which is the combination of quantitative metrics such as LOC (Lines Of codes) and ABC where A is for ASSIGNMENT which is total number of variables used in code and B is for BRANCH (Number of function calls) and C is for CONDITION (Number of if-else statements). And ABC is calculated as:

$$|ABC| = (A^2 + B^2 + C^2)^{1/2}$$

(2)

Also FP (Function points), Token Count (TC) and well known Halstead Metrics (HSS) are used. They propose this simulation is to show that complexity for same code different for different metrics. The effectiveness of any metric is different for procedural and object oriented approach. So a hybrid approach was taken to get accurate complexity value. Weighted linear mathematical evaluation is done in which 40 percentage is given to procedural metrics and rest of the 60 percentage weightage is given to object oriented metrics. And in procedural weight of 40 percent is further divided into two parts in which 20 percentage is given to quantitative metrics (Example – Halstead) and 30 percentage is given to structural metrics (Cyclomatic complexity and Myers Cyclomatic Complexity).

In [14] antipattern detection system based on metrics and rules for the object oriented software. This system consists of three main mechanisms to detect an antipattern. These mechanisms are "Metric Analyzer", "Static Code Analyzer" and "Filtering Mechanism" and also specified three antipatterns to analyze:

1. Blob
2. Swiss Army Knife and
3. Lava Flow.

R-Model is proposed in [15], to extract the software reliability at starting stage and to enhance the product quality.
As in [16] relation between design patterns and code complexity is proposed through SW Visualization. For design patterns CREATIONAL, STRUCTURAL and BEHAVIORAL patterns are taken into account and cyclomatic complexity and Coupling and Cohesion are used to detect the code complexity. Finally in [17] predicted the design quality of object oriented software using UML diagrams. The design phase is considered for the work and the establishment of relationship between software design quality and product quality is achieved and to draw UML diagrams UMLet 9.03 Under GNU tool was used. They concluded that establishing metrics play a decisive role in determining the quality of the object oriented software.

III. UML DIAGRAM

With the help of UML diagram various metrics can be evaluated and predicted the following table I gives the demonstration of design rules and if the UML is useful to extract the particular metric or not.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Design Rule</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public Attributes</td>
<td>Encapsulation</td>
<td>Yes</td>
</tr>
<tr>
<td>Public Methods</td>
<td>Narrow Interfaces</td>
<td>Yes</td>
</tr>
<tr>
<td>Method Arguments</td>
<td>Narrow Interfaces/Method Arguments</td>
<td>Yes</td>
</tr>
<tr>
<td>Directional References</td>
<td>Decoupling</td>
<td>Yes</td>
</tr>
<tr>
<td>References</td>
<td>Decoupling</td>
<td></td>
</tr>
<tr>
<td>N LOC Methods</td>
<td>Method Complexity</td>
<td>No</td>
</tr>
<tr>
<td>Weighted Method</td>
<td>Method and Class Complexity</td>
<td>No</td>
</tr>
<tr>
<td>Method Count</td>
<td>Class Complexity/1:1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Class:Abstraction/Maximize Cohesion</td>
<td></td>
</tr>
<tr>
<td>Interface Size</td>
<td>Maximize Cohesion</td>
<td>Yes</td>
</tr>
<tr>
<td>Abstract References</td>
<td>Abstract Interfaces/Decoupling</td>
<td>Yes</td>
</tr>
<tr>
<td>Un-Cohesive Methods</td>
<td>Maximize Cohesion</td>
<td>No</td>
</tr>
</tbody>
</table>

A. Relation of Quality attributes and Quality Metrics

[20] Describes the correlation between metrics and attributes depicted in table II.

<table>
<thead>
<tr>
<th>Metrics Attributes</th>
<th>WMC</th>
<th>RFC</th>
<th>LCOM</th>
<th>CBO</th>
<th>DIT</th>
<th>NOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Efficiency</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Reusability</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Testability/ Maintainability</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Less</td>
<td>Yes</td>
</tr>
<tr>
<td>Understandability</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Less</td>
</tr>
</tbody>
</table>
B. Proposed Models for Quality Assessment

The following table III gives the illustration of various models proposed for the quality assessment and also depicts that which phase is given the preference for the evaluation.

TABLE III: MODELS PROPOSED FOR THE QUALITY ASSESSMENT

<table>
<thead>
<tr>
<th>Model or Approach</th>
<th>Dominant Phase</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMOOD[2]</td>
<td>Design phase</td>
<td>Quality attribute and Design metrics</td>
</tr>
<tr>
<td>RuCMAp[3]</td>
<td>Coding Phase</td>
<td>Functions accessed at run time and object instantiated</td>
</tr>
<tr>
<td>UML &amp; FSM [12]</td>
<td>Design Phase</td>
<td>None</td>
</tr>
<tr>
<td>R-MODEL[15]</td>
<td>Design Phase</td>
<td>design metrics and reliability at the design level</td>
</tr>
<tr>
<td>SW Visualization[16]</td>
<td>Cyclomatic complexity and Coupling and Cohesion(Coding Phase)</td>
<td>Design patterns and code complexity</td>
</tr>
<tr>
<td>UML[17]</td>
<td>Design Phase</td>
<td>Software design quality and product quality</td>
</tr>
</tbody>
</table>

IV. CONCLUSIONS

This survey would help academic and research scholars and software organizations in a certain way to be able to reach at a level where it would be easier to take good decisions, and plan accordingly with better allocation of resources to reduce the cost to ensure the high quality of software. Another impact can be in evaluating the effects of programmers coding style differences in the detection of faults and bugs. Also only class diagram is effective for the metric calculation where as other UML diagrams must be in cooperated to achieve better results.

V. REFERENCES


[10] Puneet Kumar Goyal; Gamini Joshi “QMOOD metric sets to assess quality of Java program” Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014


TO CITE THIS PAPER

Available online through- http://www.ijifr.com/searchjournal.aspx