Software Quality Analysis by Object Oriented Approach in Design Phase Using UML

Anand Handa

Abstract- Software Quality is the degree to which a finished product conforms to its specifications. The earlier a fault is detected and is removed the easier it is to fix. Object oriented metrics focus on the combination of functions and data as an integrated object. Object oriented paradigm substantially improves productivity due to the effect of reuse. Requirement specifications, designs and test plans are all artifacts that could potentially be fully or partially be reused in different projects.

Object Oriented Methodology is an emerging trend in software development for scientific and business applications. Design of the solution domain has an impact on the overall quality of the software. Merging of all individual design quality metric tools as a package along with other design principles like abstraction and stability could serve the developer better as plug-ins for IDEs.

Index Terms- Solution domain, metrics, tools. Object Oriented Design, Metrics and tools.

I. INTRODUCTION

Quality is defined as the level to which a product conforms to its requirements. This implies that the requirements must be clearly and unambiguously stated in such a way that they cannot be misunderstood. Thus insuring the delivery of high-quality software is becoming an increasingly important goal in the life-cycle of software developed by serious companies.

The quality of the software depends on the design of a class. Much effort goes in repairing the bad design if identified in later stages of development process. Object Oriented Design metrics are helpful in identifying faulty design at early stage of software development. Many tools are available individually to measure a Java program before and after implementation. This paper presents the concept of merging design metric tools as a package along with other design principles like abstractness and stability. The stability of the software tool is an essential feature that adds much to the efficiency and performance of it. Many of the metrics and quality models currently available for object-oriented software analyses can be applied only after a product is complete, or nearly complete.

II. SOFTWARE QUALITY METRICS

The first step to making use of metrics should involve assessing a number of available metrics and choosing a suite for use according to a company’s quality objectives ([1] [2] [3]). There are essentially three measurable entities: products, processes, and people.

At the end of the day, if used properly, metrics allow us to quantitatively define success and failure, and/or the degree of success or failure for a product, process or person. It allows us to identify and quantify improvement, lack of improvement or degradation in the performance of a product, process or person. It helps us to make meaningful and useful managerial and technical decisions. It provides user in making quantified and meaningful estimates. It helps in making meaningful comparisons can only be made if both the similarities and dissimilarities of the products, processes or people being compared are taken into account.

III. METRICS AND OBJECT ORIENTED PARADIGM

While metrics for the traditional functional decomposition and data analysis design approach measure the design structure and/or data structure independently, object-oriented metrics must be able to focus on the combination of function and data as an integrated object. The success of the software is mainly dependent on its design. A significant number of Object Oriented design quality metrics are defined among which CK metrics (Chidamber and Kemerer) [4] are popular in the literature. The Metrics are used to calculate the efficiency parameter and various tools are used for that which are discussed in this paper and are as follows

A. Weighted Methods per Class (WMC)

Consider a class C1, with methods m1, … mn. Let cl, … cn be the static complexity of the methods.
The mathematical formula is as follows:

\[ WMC = \sum_{i=1}^{n} C_i \] (1)

WMC analyzes the class structure and the result has a bearing on the understandability, maintainability, and reusability of the system as a whole. The number of methods and the complexity involved is a predictor of how much time and effort is required to develop and maintain the class. The WMC can be calculated by using Equation (1) for any number of methods. The larger the number of methods in a class, the greater the potential impact on children, since children inherit all of the methods defined in a class. Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse.

Chidamber and Kemerer clarified the definition of method count as “the methods that require additional design effort and are defined in the class should be counted, and those that do not, should not”.

B. Depth of Inheritance Tree (DIT)

The depth of inheritance of a class is its depth in the inheritance tree. If multiple inheritance is involved, then the depth of the class is the length of the maximum path from the node representing the class to the root of the tree [5]. The root class has a DIT of 0. DIT is essentially a measure of how many ancestor classes can possibly affect this class. It is worth noting that deeper trees constitute greater design complexity, since more methods and classes are involved. However, deeper trees also signify a greater level of internal reuse in the system so a balance between reuse and reduced complexity needs to be struck. This metric primarily evaluates efficiency and reuse but also relates to understandability and testability.

C. Number of Children (NOC)

NOC simply counts the number of immediate subclasses subordinate to a particular class in the class hierarchy. This gives an indication of the potential influence a class can have on the design and on the system. The greater the number of children, the greater the likelihood of improper parent abstraction, and it may be an indication of sub-classing misuse.

Again, there has to be a compromise because a greater number of children indicate a larger degree of internal reuse of the particular class. If a class has a large number of children, it may require more testing of the methods of that class, thus increase the testing time. NOC, therefore, primarily evaluates efficiency, reusability, and testability.

D. Coupling between Object Classes (CBO):

CBO for a class is a count of the number of non-inheritance related couples with other classes. Excessive coupling between objects outside of the inheritance hierarchy is detrimental to modular design and prevents reuse since the more independent an object is, the easier it is to reuse in a different application. Also, the larger the number of couples a class has, the more sensitive it is to changes in other parts of the design thus making maintenance more difficult.

It measures the interdependency between the classes. An object of a class can use the service or object of another class. The objective is to reduce the much interdependency (cross coupling) to increase the clarity of the solution.

E. Response for a Class (RFC)

It helps to measures response set of a class. When an object of a class sends a message, the methods executed inside and outside of a class are counted. The amount of effort in debugging, testing and maintenance is depending on response count. 

\[ |RS| = |M| \cup \forall i \{R_i\} \]

Where \( \{R_i\} \) = set of methods called by method \( i \) and \( |M| \) = set of all methods in the class ([6] [7]).

IV. EXTRACTING STRUCTURAL METRIC INFORMATION FROM UML DIAGRAMS

It has been established that all the information required can be obtained from a combination of class diagrams, activity diagrams, collaboration diagrams and sequence diagrams. The calculative analysis of the various approaches can be now done with the help of the UML approaches. Calculating WMC will require the information from two types UML diagrams and various methods that have been used earlier in which object oriented approach was not used. The following methodologies help in identifying the enhancement of the OO feature.
A. Class Diagram:

The class diagram will be used for obtaining a list of methods for each class. By default, the Cyclomatic Complexity of each method will be one. However, if there are methods for which there exists an Activity Diagram describing changes in activity within the methods, the Cyclomatic Complexity for those methods should be calculated from their Activity Diagrams.

The inherited methods are not counted unless they are re-defined in the current class.

B. Activity Diagram:

Activity diagrams can be used to show the changes in activity within the methods. They are very similar to flowcharts. If a method has an activity diagram associated with it, its cyclomatic complexity is calculated as follows:

\[
\text{Cyclomatic Complexity} = \text{no. of edges} - \text{no. of nodes} + 2
\]

If a method does not have an activity diagram associated with it, then its cyclomatic complexity is taken to be 1. This follows from the notion that in theory, object-oriented methods are so small and specific that their complexity can be taken to be 1.

C. Depth of Inheritance Tree (DIT)

The depth of inheritance of a class is its depth in the inheritance tree. If multiple inheritance is involved, then the depth of the class is the length of the maximum path from the node representing the class to the root of the tree. The root class has a DIT of 0. As shown in the example below, the DIT metric is easily measured by looking at a particular class in a class diagram. The class Animal is the root class of the hierarchy shown in the example and therefore has a DIT of 0. The classes below it (DomesticAnimal, FarmAnimal, WildAnimal) have a DIT of 1 and their children in turn have a DIT of 2.

D. Number of Children (NOC)

NOC simply counts the number of immediate subclasses subordinate to a particular class in the class hierarchy. This information is easily obtainable from a class diagram as follows.

E. Coupling between Objects (CBO)

CBO for a class is a count of the number of non-inheritance related couples with other classes.

UML class diagrams will be needed to obtain information for calculating CBO. Class diagrams can show the different couplings between objects.
F. Lack of Cohesion in Methods (LCOM)

Cohesion refers to the degree of interconnectivity between attributes of a class. A class is cohesive if it cannot be further divided into subclasses. It measures the method behavior and its relevance where it is defined. Pair of methods using data object proves the cohesiveness where as the methods not participating in data access makes it less cohesive. Consider C is a class and M1, M2...Mn are its methods using set of class instances [9]. I1={a,b,c,d}, I2={a,b,c} and I3={x,y,z} are set of instances used by the methods M1, M2 and M3 respectively. If intersection of object set is non-empty then the method using them is cohesive and their relevance in the class is proved. i.e. I1 ∩ I2 = {a, b, c} means M1 and M2 are cohesive. But intersection of I1, I3 and I2, I3 is empty set. High count in LCOM shows less cohesiveness and class need to be divided to subclasses.

Apart from CK and MOOD, other metrics [8] based on Object Oriented principles are also assess the design quality.

V. COMPARATIVE STUDY

It is known from the study that very few commercial and open source tools are available for quality evaluation of object oriented software design ([10] [11]). Each tool is performing well with defined metrics individually but failed to cover all proposed in the literature. Output format of the report generated by few tools are not having friendly features which will have an impact on assessing the result.

The Table 1 shows the comparative study of the various tools that are used in the software analysis in both ways. The proposed methodology shows the gradual increase in the efficiency when design phase has been effectively measured and implemented. The use of UML increases the efficiency of the object oriented model to a greater extent.

(\text{\checkmark}) In Table 1: Signifies the existence of the mentioned factors.

VI. LITERATURE SURVEY

Since metrics are quantitative measure of a software design and implementation, a number of individual tools have been developed and made accessible as open source tools. A few commercial tools are also available for measuring process and product metrics.

A paper by P. Edith Linda et al. (2011) focused on different tools available and proposed a web page so that all the tools are accessible at the same place.

Dr. Rakesh Kumar and Gurvinder Kaur (2011) have done a comparative study on the complexity of Object Oriented Design metrics proposed by Shyam R. Chidamber, Chris F. Kemerer and Li.

Dr. M.P Thapaliyal and Garima Verma (2010) have done an empirical analysis on few metric data and their relationship with software defects.

Rudiger Lincke et al. (2008) have done analysis and comparison of the output of different tools on different projects. It is shown that the output of product metrics is dependent on tools used.

Linda Westfall, The Westfall Team (2005) defines 12 steps to useful software metrics suitable for organization. It focuses on refining the metrics for organization so that better product can be developed.

Sandeep Purao and Vijay Vaishnavi (2003) present a rigorous survey on product metrics. It focuses on understanding and classification of ongoing research in Object Oriented metrics.

Stojanovic M and El Eman K (2001) developed a tool, ES2, for collecting the design quality metrics for C++ and JAVA source code. This analyzer is basically implemented on top of Source Navigator IDE for analyzing large amount of code with cross-references and links amongst classes.
Dr. Linda H. Rosenberg focuses on Traditional and Object Oriented Metrics adapted for Object Oriented environment to evaluate the principle object oriented structures and concepts.

VII. CONCLUSION AND FUTURE ENHANCEMENT

In this paper we have made an effort to find out the nature of relationships between the Defects in software designs developed in Java and some selected metrics such as class SIZE, WMC and CBO. The object oriented approach that has been applied at the design phase level of the software project development helps in improving the efficiency of the project to a greater extent than the earlier proposed models. Also, how do all these metrics impact the defects proneness of software have been analyzed? Various studies in the past have shown a positive and significant relationship between the number of defects and SIZE, WMC and CBO. Most of these studies had used either a single project or controlled similar projects to collect data for these variables. But in this study, we tried to test these relationships to different projects and have found some inconsistencies in results. Though, class size has been found to have a positive & significant impact on defects, which was consistent with earlier findings, but other two variables i.e. WMC & CBO have been found to have insignificant impact on defects which is not consistent with earlier findings. One the possible explanation for this could be the difference in the nature of projects which have been used in our study to collect metric data related to these variables. One the important conclusions that may be drawn from this study is that even most recognized metrics that are used to predict the defect proneness of software may not be sufficient enough when applied to diverse kind of projects. We may need some more metrics to enhance the predictability in defect proneness of software designs. The ultimate goal is to develop predictive models that may be used to support decision making, e.g., decide which classes should undergo more intensive verification and validation. Regardless of the structural attribute considered, most metrics have been so far defined and collected based on a static analysis of the design or code.

REFERENCES


AUTHOR’S PROFILE

Anand Handa received his B.Tech in Computer Science & Engineering from University Institute of Engineering and Technology, Kanpur, India in 2009. He is pursuing M.Tech from RKFIST, RGPV, Bhopal (M.P.) in Computer Science & Engineering. His areas of interest are Software Engineering and Soft Computing. He has published and presented a research paper on Meta Architecture in a national conference.