The Role of Metrics in Object–Oriented Software Development : Some Issues

Anil Kumar Malviya[1], Sanjeev Patwa[2]

ABSTRACT

DeMarco(1982) once said: You can neither predict nor control what you can not measure. Werner Karl Heisenberg said: "Since the measuring device has be constructed by the observer....we have to remember that what we observe is not nature in itself, but nature exposed to our method of questioning" (in "Physics and Philosophy" (1958)). These fundamental realities underlie the importance of software metrics. The intent of Software engineering is to provide a framework for building software with higher quality. Software metrics provide a quantative basis for the development and validation of models of the software development process. Metrics can be used to improve software productivity and quality. This paper presents the current state of metrics and discusses the role of object-oriented metrics in object-oriented development.

Keywords: Measurement, software metrics, object-oriented metrics.

1 Introduction

Measurement is fundamental to any engineering discipline, and software engineering is no exception. Lord Kelvin said: "When you can measure what you are speaking about and express it in nubers, you know something about it; but when you can not measure, when you can not express it in numbers, your knowledge is of a meager and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely in your thoughts, advance to the stage of a science". Even when it is not clear how we might measure an attribute, the act of proposing such measures will open a debate that leads to greater understanding [5][18].

Software people seem to have a love-hate relationship with metrics. On one hand, they despise and distrust anything that sounds or looks like a measurement. They are quick to point out the "flaws" in the argument of anyone who talks about measuring software products, software process, and (especially) software people. On the other hand, these same people seem to have no problems identifying which programming language is the best, the stupid things that managers do to "ruin" projects, and where methodology works in what situations [1]. Skeptics claim metrics are useless and expensive exercises in pointless data collection, while proponents argue justifiably they are valuable management and engineering tools.

Measurements have been widely used in many engineering disciplines, yet in the computer software industry there still have been some doubts about their use in this field. During recent years, the interests in software metrics have grown both greatly and steadily in software industry. With the project programmer and manager focusing on software productivity and software quality, there exist needs for better technique of software development and software metrics during the process of development [20].

[1]Assistant Professor, Kamla Nehru Institute of Technology Sultanpur,U.P. anilkmalviya@yahoo.com

[2]Lecturer,Faculty of Arts,Science and Commerce Mody Institute of Technology & Science (Deemed University),Lakshmangarh, Sikar, Rajasthan sanjeevpatwa@yahoo.com

We first briefly describe "What is measurement?", "What are Software Engineering Metrics?", followed by "Current state of Software Metrics".

1.1 What is Measurement?

Measurement is defined as 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 two broad types of measurement: direct and indirect, Direct measurement of an attribute is measurement which does not defined on the measurement of any other attribute, indirect measurement of an attribute is measurement which involves the measurement of one or more other attributes [7].

1.2 What are Software Engineering Metrics?

Software metrics are units of measurements. The term "metrics" is also frequently used to mean a specific measurement taken on a particular item or process. Software metrics are quantifiable measures that could be used to measure different characteristics of a software system or a software development process. Software metrics can be classified into three categories, product metrics, process metrics and project metrics. Product metrics are used to quantify characteristics of the product being developed i.e. the software. Examples include the product size, complexity, design features, performance and quality level. Process metrics are used to quantify characteristics of the process being used to develop the software. Examples include the effectiveness of defect removal during development, the pattern of testing defect arrival, and the response time of the fix process. Project metrics are those that describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule and productivity ([1], [11], [12]).

1.3 Current State of Software Metrics

The current state of software metrics is not very satisfactory. In the past, many metrics and a number of process models have been proposed. Unfortunately, most of the metrics defined have locked one or both of two important characteristics: a sound conceptual, theoretical basis and statistically significant experimental validation. Most of these have been defined and then tested only in a limited environment, if at all. In some cases, remarkable success has been reported in the initial application or validation of these metrics. However subsequent attempts to test or use the metrics in other situations have yielded very different results. One part of the problem is that we have failed to identify a commonly accepted set of software properties to be measured. As a result, the same metric has been used to measure very different software properties [2]. Despite these problems, software metrics in limited environment can significantly in improving software quality and productivity. Recent studies indicate that there are strong relationship between OO metrics and software quality ([3], [6], [13], [19] and [17]). The rest of the paper has been organized as follows. In the next section we provide the traditional metrics. In section 3 we discuss the object oriented metrics. Section 4 describes the case studies of object oriented software engineering metrics. We conclude the paper in section 5.

2 Traditional Metrics

A large number of traditional metrics have been proposed to measure the complexity metrics are the McCabe's Cyclomatic metrics [19] and Helstead's Software science metrics. McCab has proposed a graph-theoretic complexity measure that is widely accepted, probably it is easily calculated and is intuitively satisfying. Helstead software science is based on a refinement of measuring program size by counting lines of code. It is one of the most widely accepted measures in industry and universities and has been supported by several empirical studies. Helstead software science can save millions in maintenance cost, but while current measures can be used to some degree, most are not sufficiently sensitive to comprehensive ([2],[10]).

Complexity metrics can be used to predict critical information about reliability and maintainability of software system. Complexity metrics also provide feedback during the software project to help control the design. During testing and maintenance, they provide detailed information about software modules to help pinpoint areas of potential instability [18].

3 Object Oriented Metrics

The Object Oriented approach and metrics are getting a lot of attention from software development community. This is due to a variety of claims by many software researchers and practitioners that an object oriented approach to software development leads to better productivity, reliability, maintainability, software reusability and increased extensibility. In order to achieve these goals, a variety of object oriented software metrics have been proposed in literature to help track the status of software development. In subsection 3.1, we define the Chidamber and Kamerer (CK) metrics suite. The MOOD set of object oriented are given in subsection 3.2. Subsection 3.3 defines the metrics proposed by Lorenz and Kidd.

3.1 Object Oriented Metrics proposed by Chidamber and Kamerer

Chidamber and Kamerer [5] have developed one of the most widely referenced set of class based metrics for Object Oriented systems. This metrics suite for OOD is the deepest research in OO metrics investigation. The Six metrics they have identified are listed below.

- Weighted Methods per class (WMC): WMC measure the overall complexity of the class. It is the sum of the complexities of the methods implemented within a class. Method complexity of each class is assigned 1 the number of methods per class is a measure of WMC. The number of methods and the complexity of the method involved is a predictor of how much time and effort is required to develop and maintain the class. Classes with large number of methods are likely to be more application specific, limiting the possibility of reuse [5].

  WMC= number of methods defined in class

- Depth of the Inheritance Tree (DIT): DIT of a class in an inheritance hierarchy is the maximum length from the class node to the root of the tree. The deeper the class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behavior [5].

  DIT=maximum inheritance path from the class to the root class

- Number of Children (NOC): NOC counts the number of classes which inherit from a particular class (i.e. the number of classes in the inheritance tree down from a class). Greater the number of children, greater the reuse, since inheritance is a form of reuse. Greater the number of children, the greater the likelihood of improper abstraction of the present class. If a class has a large number of children, it may be a case of misuse of sub classing. The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class.

  NOC=number of immediate subclasses of a class

- Coupling between Objects (CBO): CBO is defined as the total number of other classes to which class is coupled. A class is coupled to another class if it uses the member method and /or instance variable of the other class. Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier is to reuse it in another abstraction.

  CBO=number of classes to which a class is coupled

- Response for a class (RFC): RFC is the count of the set of all methods that can potentially be invoked in response to all methods accessible within to all methods accessible within the class hierarchy. The larger the number of methods that can be invoked from a class, the greater the complexity of the class.

  RFC=|RS| where RS is the response set for the class.

- Lack of Cohesion in methods (LCOM): LCOM is the count of the number of method pairs whose similarity is 0 minus the count of method pairs whose similarity is not zero. The larger the number of similar methods, the more cohesive the class is. Cohesiveness of methods within a class is desirable, since it promotes encapsulation and lack of cohesion implies classes should probably be split into two or more subclasses. Low cohesion increases the likelihood of errors during the development process.

3.2 MOOD Set of Object Oriented Design Metrics

The MOOD metrics set refers to a basic structural mechanism of the OO paradigm as encapsulation (MHF and AHF), inheritance (MIF and AIF), polymorphism (PF), and message passing (CF) and are expressed as quotients. The Six MOOD metrics are listed below.

- Method Hiding Factor (MHF)

  MHF is defined as the ratio of the sum of the invisibilities of all methods defined in all classes to the total number of methods defined in the system under consideration. The invisibility of a method is the percentage of the total classes from which this method is not visible.

- Attribute hiding factor (AHF)

  AHF is defined as the ratio of the sum of the invisibilities of all attributes defined in all classes to the total number of attributer defined in the system under consideration.

- Method Inheritance factor (MIF)

  MIF is defined as the ratio of the sum of the inherited methods in all classes of the system under consideration to the total number of available methods (locally defined plus inherited) for all classes.

- Attribute Inheritance Factor (AIF)

  AIF is defined as the ratio of the sum of the inherited attributes in all classes of the system under consideration to the total number of available attributes (locally defined plus number of all classes).

- Polymorphism factor (PF)

  PF is defined as the ratio of the actual number of possible different polymorphic situations for class C_i to the maximum number of possible distinct polymorphic situations for class C_i.

- Coupling Factor (CF)

  CF is defined as the ratio of the maximum possible number of couplings in the system to the actual number of coupling not imputable to inheritance.

3.3 Object Oriented Metrics proposed by Lorenz and Kidd:

Lorenz and Kidd divide class based metrics into four broad categories: size, inheritance, internals and externals. The metrics proposed by Lorenz and Kidd are listed below [18].

- Class size : The overall size of a class can be determined using the following measures:

1. the total number of operations (both inherited and private instance operation) that are encapsulated within the class
2. the number of attributes (both inherited and private instance attributes) that are encapsulated by the class

- Number of Operation Overridden by a Subclass (NOO): There are instances when a subclass replaces an instruction inherited from its superclass with a specialized version for its own use, this is called overriding. Large values for NOO generally indicate a design problem.
- Number of operations added by a subclass (NOA): Subclasses are specialized by adding private operations and attributes. As the value for NOA increases, the subclass drifts away from the abstraction implied by the superclass.
- Specialization Index (SI): This is the measure of the degree of specialization for each of the subclass in an OO system.

4. Case Studies of Object Oriented Software Metrics

We will break our look at case studies into the following areas:

4.1 anecdotal metric information
4.2 the General Electric Report
4.3. Chidamber and Kamerer's research
4.4 Lorenz's Research
4.5 Edward V. Berard Work
4.6 Harrison,Counsell and Nithi Work
4.7 Some other observations on Object Oriented Metrics
4.8 Our Research Work

4.1 Anecdotal metric information

Anecdotal Object-Oriented Software engineering metrics information includes [1]:

- It takes the average software engineering about 6 months to become comfortable wit object-oriented technology
- The average number of lines of code per method is small, i.e., typically 1-3 line code, and seldom more than 10 lines of code
- The learning time for smalltalk seems to be on the order of two months for an experienced programmer
- Once a programmer understands a given Object-Oriented programming language, He/She should plan on taking one day per class to (eventually) understand all the classes of the library.
- Object-Oriented technology yields higher productivity, e.g. fewer software engineering accomplishing more work when compared to traditional teams.

4.2 The General Electric Report

Deborth Boehm-Davis and LyleRoss conducted a study for General Electric (in 1984): comparing several development approaches for Ada software (i.e. structured analysis/structured design, Object Oriented design (Booch), and Jackson System Development). They found that the object-oriented solutions, when compared to the other solutions [1]:

- Were simpler (using Maccab's and Halstead's metrics)
- Were smaller (using lines of code as metric)
- Appeared to be better suited to real time applications
- Took less time to develop

4.3 Chidamber and Kamerer's research

One of the first suite of OO design measures was proposed by Chidamber and Kamerer as discussed in section 3.1.The Chidamber and Kamerer [5] identified six Object Oriented Metrics. An automated data collection tool was then developed and implemented to collect an empirical sample of these metrics at two field sites in order to demonstrate their feasibility and suggest ways in which managers may use these metrics for process improvement. Some observations are as follows:

- All the metrics satisfy the majority of the properties prescribed by Weyuker, with one strong exception, property 6.Property 6 is not met by any of the metrics in this suite.
- A benefit of having a suite of metrics is that there is the potential for multiple measures of same underlying construct.
- By using the metrics suite they can identify areas of the application that may require more rigorous testing and areas that are candidates for redesign.

4.4 Lorenz's Research :

Mark Lorenz and Jeff Kidd have published the results of their object-oriented software engineering metrics work. Some of the interesting item in there empirical data include:

- The ratio of key (important) classes to support classes seems to be typically 1 to 2 and user interface intensive applications tend to have many more support classes.
- The average number of person day to develop a class is much higher with C++ than is with smalltalk, e.r., 10 days per smalltalk class and 20 to 30 days per C++ class.
- The higher the number of lines of code per method,the less object-oriented the class is.
- SmallTalk applications appear to have a much lower average number of instance variable per class when compared to C++ applications.

4.5. Edvard V.Berard work

Berard,E.V. [1] has worked on object-oriented project since 1983.Some observations from some of the projects include:

1. On a very large (over 1,000,000 lines of code) object-oriented project, all of the source code was run through a programe that reported on the metrics for that software. Some observations are:
   - Over 90% of all the methods in all of the classes had fewer than 40 lines of (carriage return)
   - Over 95% of all the methods had a cyclometic complexity of 4 or less.
2. On a small project (about 25,000 lines of code), staffed by 3 software engineering working half-time on the project:
   - The project was completed in six calendar months, i.e., a total of 9 software engineering expended.
   - When the code was first compiled and compilation errors were found, and no more errors were found, before the code was delivered to the customer.

4.6. Harrison, Counsell and Nithi Work:

Harrison,R.,Councell,S.J. and Nithi, R.V. describe the result of an investigation into a set of metrics for object oriented design called the MOOD metrics. Result shoe that the metric could be used to provide an overall assessment of a software system, which may be helpful to managers of software development projects [9]. Some observations are

- Six MOOD metrics has led us to conclude that, as far as information hiding,inheritance,coupling, and dynamic binding are concerned the six MOOD metrics can be shown to be valid measure within the context of theoretical frame work (specific)
- Empirical results indicate that the MOOD metrics, Operate at the system level
- The MOOD metric could be of use to project managers, providing an Overall assessment of a system.
- However their utility will continue to be questioned until a sufficient number of empirical validations have been performed at a system level to establish casual relationships between the metrics and external quality attributes of systems, such as reliability, maintainability, testability etc.

4.7 Some Other Observations on Object-Oriented metrics

- Briand,L.C. et al. [12] investigate existing object-oriented coupling, cohesion and inheritance measures and the probability of fault detection in system classes during testing and their capability to predict where faults are located. The results still support the idea that the measurement of OO designs can still shed light on their quality.
- Li,W. et al. [13] concentrate on several object-oriented software metrics and the validation of these metrics with maintenance effort in two commercial systems. The results of the analyses of the two object-oriented systems show that-

1. There is a strong relationship between metrics and maintenance effort in object-oriented systems.
2. Maintenance effort can be predicted from combinations of metrics collected from source code.
3. The prediction is successfully cross validated.

- Mayer,B. [15] presents a classification of software metrics and five basic rules for their application.
- Subramanyam, R. et al. [19]provide empirical evidence supporting the role of OO design complexity metrics, specifically a subset of the Chidamber and Kemerer suite, in determining software defects. Some observations of the research are:

1. After controlling the size, some of the measures in the CK suite of OO design complexity metrics significantly explain variance in defects.
2. The effects of certain OO design complexity metrics, such as WMC,CBO and DIT, on defects were found to differ across the C++ and Java samples in the study.
3. The programming language might play a role in the relationship between OO design metrics and defects.

- Cartwright,M.et al. describe an empirical investigation into an industrial object-oriented (OO) system comprised of 133,000 lines of C++. Some findings of the research are:

1. Chidamber and Kemerer's DIT and NOC metrics could therefore be used to pinpoint classes that are likely to have higher defects densities.
2. Two other measures, available by the design stage, the number of events for a class (EVNT) and the number of states for a class (STATES), can be useful

and a accurate predictions of the number of defects and LOC.

3. LCOM to be "adequate" predictors of fault-prone classes

- The Software Assurance Technology (SATC) at NASA Goddard Space Flight Centre [17] has found that a combination of "traditional" metrics and metrics that measure structures unique to object oriented development is most effective.
- Dutoit, A.H. et al. [6] describe three proof-of-concept experiments to illustrate the value of communication metrics in software development projects and propose a statistical framework based on structural equations for validating these communication metrics.
- Chidamber,S.R. et al. [4] suggest that the OO metrics provide significant explanatory power for variations in economic variables, over that provided by traditional measures. Such as size in lines of code and after controlling for the effect of individual developers. Some observations of the empirical results across the three financial services application are as follows:

1. Useful metrics data could be called on systems that were written in a variety of programming languages and on a system that was not yet coded.
2. None of three applications at this site showed significant use of inheritance-DIT and NOC tended to have minimal values.
3. WMC,CBO and RFC tended to be highly correlated.
4. High levels of coupling (HICBO) and lack of cohesion (HILCOM) were associated with:
   - lower productivity
   - greater rework
   - greater design effort

These results showed be of interest to both practitioners and the OO research community as to the degree to which the metrics are of practical significant.

4.7 Our Research Work:

1. We have proposed model [14] that captures the relationship between the number of initial faults present in an object and some metrics defined for an object oriented software system. The proposed model is as follows:

   No.of faults=K * (DIT * WMC *RFC * CBC * LCOM)/NOC ....(1)

   The constant K will have to be worked out for specific software team of concerned organization based on experience of team members and other characteristics related to software processes. The "number of initial faults" serves as an important parameter of reliability models for software [14].

2. We propose model [14] that prepare the background for development of metrics for assessment of maintainability of software systems. The maintainability model is as follows:

   M_b=K / (DIT * CBO *RFC * LCOM * NOD)...........(2)

   The constant K will have to be worked out for specific software teams of concerned organization based on experience of team members and other characteristics related to software processes. The maintainability values of all classes of the software system can be used to find average (or mean) maintainability values of classes in the system.

   A_vM_b="M_{bi}/n ....(3)

3. In paper [14] we have presented a three level approach to measure the maintainability of a class in term of import coupling metrics. The proposed model of maintainability at analysis, high level and low level design phases given as follows:

   M_b (c)=K (1/(CBO (c) * CBO (c) * DAC'(c) * CA (c)))........(4)

   M_b(c)=K/(RFC(c)*CM(c)) ...(5)

   M_b (c)=K/((MPC(c) * ICP(m) * MM(c,d)) ....(6)

The K is constant. This model has provided a tentative measure to control the maintainability of a class at analysis, high level and low level design phase of object oriented software development. It will provide a basis for measuring software maintainability of an object oriented system in more meaningful way.

5. Conclusions

Object oriented metrics exist and do provide valuable information to object oriented developers and project managers. The SATC [17] has found that a combination of "traditional" metrics and metrics that measure structure unique to object oriented development is most effective. At this time there are no clear interpretation guidelines for these metrics although there are guidelines based on common sense and experience. Object oriented metrics technology has become an important and meaningful research field in software industry. it can aid the software project managers and developers to enhance the quality of the software development process. They can identify areas of the application that may require more rigorous testing and areas that are candidates for redesign. Using the metrics in this manner, potential flaws and other leverage points in the design can be identified and dealt with earlier in the design-develop-test-maintenance cycle of an application.

References

1. Berard,E.V., *"Metrics for Object-Oriented Software Engineering"*, The Object Agency, Inc., Internet.

2. Bieman, J.M., *"Metric Development for Object-Oriented Software"*, Internet.

3. Briand,L.C., Wust, J, Daly, J.W. and Perter, *"Exploring the relationships between design measures and software quality in Object-Oriented systems"*, The Journal of Systems and Software 51(2000) 245-273.

4. Chidamber,S.R., *"Darcy,D.P. and Kemerer,C.F.," Managerial Use of Metrics for Object-Oriented Software:An Exploratory Analysis"*, IEEE Transactions on Software Engineering, Vol.24,No.8,August 1998.

5. Chidamber,S.R. and Kemerer, C.F., *"A metric Suite for Object Oriented Design,"* IEEE Transactions on Software Engineering,Vol.20,No.6, June 1994.

6. Dutoit, A.H. and Brugge,B., *"Communication Metrics for Software Development"*, IEEE Transactions on Software Engineering,Vol.29, No.4, April 2003.

7. Fenton,N.,"Software Measurement: *"A necessary Scientific Basis"*, IEEE Transactions on Software Engineering, Vol.20,No.3,March 1994.

8. Fenton,N.E. and Pfleeger,S.L., *"Software Metrics: A Rigorious and Practical Approach"*, PWS Publishing Company.

9. Harrison,R.,Counsell, S.J. and Nithi, R.V., *"An Evalution of the Mood set of Object-Oriented Software metrics"*, IEEE Transactions on Software Engineering,Vol.24,No.6,June 1998.

10. Harrison,W., Magel, K.,Kluczny,R. and DeKock,A., *"Applying complexity metrics to program maintenance"*, IEEE Computer December 1982.

11. Jalote,P., *"An Integrated Approach to Software Engineering"*, Second Edition, Narosa Publishing.

12. Kan,S.H., *"Metrics and Models in Software Quality Engineering"*, Pearson Education Asia,2002.

13. Li,W. and Henry, S., *"Object-Oriented Metrics that predict maintainability"*, J.Systema Software, 1993,23:111-122.

14. Malviya, A.K.,Some Observation on Reliability Metrics for Object-Oriented Software", Proceeding of ICSE-2000 25-26 March, 2000, I.T., Banaras Hindu University.

15. Meyer,B., *"The role of Object-Oriented Metrics"*, IEEE Computer Nov. 1998, P.123-125.

16. McCABE, T.J., *"A complexity Measure"*, IEEE Transactions on Software Engineering,Vol SE-2,No.4, December 1976.

17. NASA,SATC, *"Applying and Interpreting Object-Oriented Metrics"* Internet

18. Pressman,R.S., *"Software Engineering: A Practitioner's Approach"*, Fourth Edition, The McGraw-Hill Companies, Inc.

19. Subramanyam, R. and Kishnan, *"Empirical Analysis of CK Metrics for Object-Oriented design Comlexity: Implications of Software Defects"*, IEEE Transactions on Software Engineering,Vol.29,No.4, April 2003.

20. Xie,T.,Huang,H.,Chen,X.,Mei,H. and Yang,F., *"Object-Oriented Software Metrics Technology"*, Technical Report, October,1999 Internet.

Author's biography

Dr. Anil Kumar Malviya is an Assistant Professor in Computer Science & Engineering Department at Kamla Nehru Institute of Technology, (KNIT), Sultanpur. He received his B.Sc. & M.Sc. both in Computer Science from Banaras Hindu University, Varanasi respectively in 1991 and 1993 and Ph.D. degree in Computer Science from Dr. B.R. Ambedkar University, Agra in 2006. He is Life Member of CSI, India. He has published over 8 papers in National & International Journals and 5 papers are in International/ National conferences and seminars. His research interests are Data mining and Software engineering, Cryptography & Network Security.

Mr. Sanjeev Patwa is Lecturer in Computer Science at FASC, Mody Institute of Technology & Science, (MITS), Lakshmangarh (Sikar) .He has long teaching experience. He received his B.Sc. (PCM) from M.L.Sukhadia University Udaipur , MSc (C.S.) & M.C.A. from JRN Deemed University, Udaipur. He also received "B" Level certificate from DOEACC Society, New Delhi. He has also done PGDCA from Rajasthan University, Jaipur. He has published some papers in National & International Journals and 5 papers are in International/National conferences and seminars. He is also author of 4 computer science books. His research interest is Software Engineering.