Classification of Software Projects Based on Software Metrics: A Review

Kunal Chopra[1], Monika Sachdeva[2] and Sunil Dhawan[3]
[1]*Department of Computer Science & Engineering,*
Shaheed Bhagat Singh State Technical Campus, Firozpur, India
[2,3]*Department of Computer Science,*
NIMS University, Jaipur, Rajasthan, India
E-mail: [1]kunalraichopra@gmail.com, [2]monika.sal@rediffmail.com,
[3]sunildhawan007@gmail.com

***Abstract*—Software metrics are developed and used by the many software organizations for the evaluation and confirmation of good software code, working and maintenance of the software product. Software metrics measure and identify various types of software complexities such as size metrics, control flow metrics and data flow metrics. Such observed and calculated software complexities should be continuously calculated, understood and controlled. One of the significant objective of software metrics is that it is applicable to both a process and product metrics. It is always a clear fact that the high degree of complexity in modules is bad in comparison to a low degree of complexity in modules. Metrics can be used in different phases of the software development lifecycle. This paper reviews the theory, called "software complexity metrics for evaluation of software projects", and analysis can be done based on various static and dynamic parameters. We will try to evaluate and analyze various aspects of software metrics in determining the quality and improvise the working of the software product development.**

Keywords: Software Metrics, Lines of Code, Control Flow Metrics, Structural Testing

I. INTRODUCTION

"Metrics don't solve problem, people solve problem, Metrics provide information so that people can solve problems."[1]

METRICES: Metrics was introduced in the context of software measurement which has become an essential tool for the good software engineering. So Metric can be defined as a quantitative measure of degree to which a system, component or process possesses a given attribute. "A handle or guess about a given attribute" for example "Number of errors found per person hours expended."

A. Need and Importance of Software Metrics

Why do we measure???...

- Determine the quality of the current product or process.
- Predict qualities of a product/process.
- Improve quality of a product/process.

Many of the best software developers measure the characteristics of the software to get some sense that whether the requirements are consistent or complete, whether the design is of high quality whether the code is ready to be tested. So basically software engineers measures the attributes of the product so as to estimate the project will be ready for the delivery or the budget may perhaps be exceeded. [2]

Measurement is a necessary practice for understanding, improving and controlling our environment. It requires rigor and care and though it has huge impact on software engineering. Measurement need is directly related to goals we set and question we ask while developing a software.

Thus Measure can defined as ***a*** quantitative indication of extent, amount, dimension, capacity, or size of some attribute of a product or process for example Number of errors in the entire code as it said that, "When you can measure the product you are talking numbers that means we can easily count or express an entity in figures similarly but when we cannot measure the product or an entity we cannot express it into numbers our knowledge is unsatisfactory type it may perhaps be the beginning of the knowledge."

B. Need of Measurement in Everyday Life

Measurement is essential in our daily lives and measuring has become the necessity and well accepted fact. It exists in the heart of many systems that governs our lives.

Economic measurements determines price and pay increases. In Medical sciences measurements help doctors to diagnose certain illness. Measurements related to atmospheric systems are the basis for whether forecasting. Without measurement, technology cannot operate. Measurement perhaps may not solely the domain of professional technologists. Everyone of us uses it in our daily life.

For instance, while making a trip we do measure distance, selects our route, measure our speed and predict the time when we arrive at our destination. So measurement helps us to understand our world, interact with our surroundings and improve our lives.

C. What is Measurement?

Measurement can be well predicted with a example in a shop if we compare the price of one commodity with another. In a garment shop we contrast sizes. And in case of journey we can compare distance travelled to

distance remained. So we can make the calculations and predictions accurate according to well defined set of rules. [3]

So measurement may be defined as the process by which the symbols and numbers are assigned to attributes of entities in the real world in such a way so as to describe them according to clearly defined set of rules.

The measurement provides information about attributes of entities. An entity can be an object or an event in real world. So an entity can be well descrived by identifying characteristics that are important to us in distinguishing one entity from another. An attribute is the feature or property of entity.

So when we describe these entities by using their attributes, we define those attributes by using numbers and symbols. Thus the price is designated as the rupees or dollars sterling, while height is defined in terms of inches and feet. Those numbers and symbols are abstractions that we usually use to reflect people's perception in the real world.

Thus measurement can be determined as the process whose definition is not accurate. To understand what measurement is we may have to ask host of questions which may difficult to answer.

1. Height of a person is commonly known attribute that can be measured. But other attributes of people, such as intelligence creates a fuss.
2. Height is commonly measured in terms of meters, inches and feet. These different scales measure the same attribute. But we can also measure height in terms of miles and more appropriate for the measurement of distance of satellite above earth but not for the measurement of the height of the person which again makes measurement definition far from accurate.
3. The accuracy of the measurement depends upon the measuring instrument and the definition of the measurement. For example length can measured with accuracy as long as the ruler is accurate and used in proper way.
4. Once we attain measurements for different aspects of real world, we need to analyze them and define conclusions about the entities from which they were derived. It also requires that what sort of changes or manipulations can we apply for the results of measurement? For example why it is acceptable to say that Joe is twice as tall as Fred but not acceptable to say that it is twice as hot today as it was yesterday?

D. Making things Measurable

"What is not measurable, make it measurable."

The above stanza suggests that one of the aims of science is to find ways to measure attributes of the things in which we are interested. Measurement makes concepts more clear and therefore more understandable and controllable. Thus, as scientists, we should find out ways to measure world; where we can already measure, we can make our measurement better.

To improve the implementation of measurement in software engineering, we need not to restrict type of measurements we make. Really measuring the un-measurable should improve our understanding of particular entities and attributes, and making software engineering as powerful as other engineering disciplines.

Strictly speaking, there are two kind of quantification: measurement and calculation.

Measurement is termed as direct quantification, as in the measurement of the height of a tree or the weight of a shipment of bricks. On the other hand calculation is indirect quantification, where we take measurements and combine them into a quantified item that describe the attribute whose value is to be determined.

For instance, when a city inspectors assign a valuation to a house, they calculate it by using certain formula that combines variety of factors which includes number of rooms, the overall floor space and the type of heating and cooling. Thu the valuation is termed to be quantification, not a measurement, and it expression as a number makes it more useful than qualitative assessment alone.

So eventually, it is necessary to modify our surrounding or our practices in order to measure something new or in an innovative way. It can be achieved by using a new tool, adding a new steps in a process, or using a new method. In many cases, change is difficult for people to accept, there are management issues to be considered whenever a management program is implemented or changed.

E. Measurement in Software Engineering

We have seen the importance of measurement in our daily life, measurement has become an essential and well accepted attribute of life. In this section, we will see instances of software engineering to see why measurement is needed.

Software Engineering briefs the collection of techniques that apply an engineering approach to the construction and maintenance of software products. It includes activities like managing, costing, planning, modeling, analyzing, specifying, implementing, testing and maintaining.

In engineering we try to impend each activity to be well understood and maintained so that there are fewer surprises as the software is designed, specified, built and maintained. On the another hand computer science gives the theoretical foundations for building software, software engineering focuses on implementing the software in a controlled and specific manner.

The significance of software engineering cannot be understood, since software pervades our lives. From

banking transactions to air traffic control, from oven controls to air bags,, and sophisticated power plants to sophisticated weapons, our life and quality of life depends upon software.

In software engineering we use various software models and theories for example in making an electrical circuit we appeal to theories like Ohms Law which gives the relation between resistance, current and voltage in the circuit. Once the scientific method suggests the validity of the subject concern, the measurement or the truth of the story, we continue to use measurement to apply the theory to practice. Thus to build a circuit with a specific current and resistance, we know what voltage is required and we use instruments to measure that we have such voltage in the given battery.

It is difficult to predict the mechanical, electrical and civil engineering without a central for measurement. Indeed science and engineering can neither be effective nor practical without measurement. But measurement in software engineering has been considered a luxury. For most development projects:

1. Gilbs principle of Fuzzy Targets: projects without clear goals will not achieve their goals clearly. For example we promise to make a reliable, user-friendly and maintainable without specifying clearly and objectively what these terms mean.
2. We do not quantify or predict the quality of products we produce. Thus we cannot tell a potential user how reliable the product will be in terms of likelihood of failure in a given period of use, or how much work will be needed to port the product to a different machine environment.

Since measurements are made they are often done inconsistently, infrequently, inconsistently and incompletely. The incompleteness can be frustrating to those who really want to use the results. For instance, a developer may claim that 80% of all software costs involve maintenance, or that there on average 55 faults in every 1000 lines of software code. But we are not always told how these results were obtained, how experiments were designed and executed, which entities were measured and how, and what were realistic error margins. Without this additional information, we remain skeptical and unable to decide whether to apply results to our own situations. [4]

The Software complexity is based on well-known software metrics, this would be likely to reduce the time spent and cost estimation in the testing phase of the software development life cycle (SDLC), which can only be used after program coding is done. Improving quality of software is quantitative measure of the quality of source code.

This can be achieved through definition of metrics, values can be calculated by analyzing source code or program is coded. A number of software metrics widely used in the software industry are still not well understood.

Although some software complexity measures were proposed over thirty years ago and some others proposed later. Sometimes software growth is usually considered in terms of complexity of source code.

Various metrics are used, which unable to compare approaches and results. In addition, it is not possible or equally easy to evaluate for a given source code.

Software complexity, deals with how difficult a program is to comprehend and work with Software maintainability, is the degree to which characteristics that hamper software maintenance are present and determined by software complexity.

II. THE SCOPE OF SOFTWARE METRICS

Software metrics include many activities that may include some sort of measurement. It may help in determining various activities like:

1. Structural and complexity metrics.
2. Management by metrics.
3. Evaluation of methods and tools.
4. Cost and effort estimation.
5. Productivity measures and models.
6. Data collection.
7. Reliability models.
8. Quality modals and measures.
9. Performance evaluation and models.

A. Why do We Need to Classify

From software engineering point of view software development experience shows, that it is difficult to set measurable targets when developing software products.

Produced/developed software has to be testable, reliable and maintainable. On the other side, “You cannot control what you cannot measure”. [5]

In software engineering field during software process, developers do not know if what they are developing is correct and guidance are needed to help them accustom more improvement. Software metrics are facilitating to track software enhancement. Various industries dedicated to develop software, and use software metrics in a regular basis. Some of them have produced their own standards of software measurement, so the use of software metrics is totally depending upon industry to industry. In this regards, what to measure is classified into two categories, such that software process or software product.

But ultimately, main goal of this measure is customer satisfaction not only at delivery, but through the whole development process. [6]

Various software metrics have been discovered and proposed by the researchers if we take a glimpse of the history of software metrics. The software metrics range through size, design and complexities proposed by

McCabb (1976), Helstead (1977), Lorenz (1993) and Chidamber and Kermer (1994) were chosen for the improvisation in design and development of the software projects. The discovered metrics domains were non OO and OO designing in software engineering which were implemented empirically onto various software projects so as to increase the productivity and quality of the project.

Huge budget is being spent in the maintenance and improving the quality of software projects based on the criteria set by the proposed metrics. But this mechanism is somewhat not good in essence that these approaches are implemented during the maintenance phase or rarely at the design phase. This can be prevented if we classify our software projects in accordance with the software metrics.

Based on non-OO and OO design metrics we can broadly classify our software projects in the following category:

a. Size based projects.
b. Design oriented projects.
c. Approach based projects.
d. Program weakness.
e. Failures.
f. Functionality.
g. Complexity.
h. Dependency.

III. Types of Sotware Metrics

As we have discussed earlier that first obligation of any software measurement activity is identifying the attributes and entities we wish to measure. In software there two such classes:

- **Processes Metrics** are collection of software related activities.
- **Products Metrics** documents or deliverables that result from a process activity.

A. Software Process Metrics

Software process metrics involves measuring of properties of the development process and also known as management metrics. These metrics include the cost, effort, reuse, methodology, and advancement metrics. Also determine the size, time and number of errors found during testing phase of the SDLC.

B. Software Product Metrics

Software process metrics involves measuring the properties of the software and also known as quality metrics. These metrics include the reliability, usability, functionality, performance, efficacy, portability, reusability, cost, size, complexity, and style metrics. These metrics measure the complexity of the software design, size or documentation created.

C. Size Metrics: Lines of Code

Certain size metrics were proposed for measuring the software like LOC(Lines of Code), KLOC (1000 Lines of Code), SLOC(Statement Lines of Code). Lines of code is actually count of instruction statements. It's count is usually for executable statements. [7] Since the LOC count gives the program size and complexity, it is not a surprise that the more lines of code there are in a program, the more defects are expected. More surprisingly, researchers found that defect density(defects per KLOC) is also significantly related to LOC count. Previous studies pointed to a negative relationship: the larger the module size, the smaller the defect rate. For example, Basili and Perricone (1984) examined FORTRAN modules with fewer than 200 lines of code for the most part and found higher defect density in the smaller modules. Shen and colleagues (1985) studied software written in Pascal, PL/S and Assembly language and found an inverse relationship existed upto about 500 lines. Since larger modules are generally more complex, a lower defect rate is somewhat counterintuitive.

D. Halstead Complexity Metric (1977)

It distinguishes software science from computer science. According to computer science a computer program is a collection of tokens that can be classified as either operators of operands. [9] The primitive measures of Halstead's software science are:

n1 = Number of distinct operators in a program
n2 = Number of distinct operands in a program
N1 = Number of operator occurrences
N2 = Number of operand occurrences

Given the attribute measures based on that, Halstead developed a system of equations which expresses the overall program length, the potential minimum volume for an algorithm, total vocabulary, the, the actual volume(the number of bits required to specify a program), the program level (a measure of software complexity), program difficulty, and other features such as development effort and projected number of faults in the software. Halstead major equations include the following:

a. Program Length (N) = $N1+N2$
b. Program Vocabulary (n) = $n1+n2$
c. Volume of a Program (V) = N*log2n
d. Potential Volume of a Program (V*)=$(2+n2)\log2(2+n2)$
e. Program Level (L) = $L=V^*/V$
f. Program Difficulty (D) = $1/L$
g. Estimated Program Length (N) = $n1\log2n1+n2\log2n2$
h. Estimated Program Level (L) = $2n2/(n1N2)$
i. Estimated Difficulty (D) = $1/L = n1N2/2n2$

j. Effort (E) = V/L = $V*D$ = ($n1$ x $N2$) / $2n2$
k. Time (T) = E/S

["*S*" is Stroud number (given by John Stroud), the constant "*S*" represents the speed of a programmer. The value "*S*" is 18]

One major weakness of this complexity is that they do notmeasure control flow complexity and difficult to compute during fast and easy computation.

E. McCabe Cyclomatic Complexity by (1976)

It was designed to indicate a programs testability and understandability. It is the classical graph theory cyclomatic number, indicating the number of regions in the graph. As applied to the software, it is the number of linearly independent paths that comprise the program. The M is equal to the number of binary decisions plus 1. [8]

If all the decisions are not binary, a three way decision can be counted as two binary decisions and n-way case statement is counted as n-1 binary decisions. The cyclomatic complexity metric is additive. The complexities of several graphs considered as a group is equal to the sum of individual graphs complexities. The general formula to compute the cyclomatic complexity is:

M= V(G) = e – n + 2p where.
V(G) = Cyclomatic number of G.
e = Number of edges.
n = Number of nodes.
p = Number of unconnected parts of the graph.

We can compute the number of binary node (predicate), by the following equation.

$V(G)= p+1$
where, $V(G)$= Cyclomatic Complexity
P= number of nodes or predicates.

The problem with McCabb's Complexity is that, it fails to distinguish between different conditional statements (control flow structures). Also does not consider nesting level ofvarious control flow structures.

F. Design Metrics

In 1994 Chidamber and Kermer proposed six OO design and complexity metrics, which became the commonly referred to CK metric suite:

1. *Weighted Method per Class* (WMC): WMC is the sum of the complexities of the methods, whereas complexity is measured by cyclomatic complexity. If one consider all the methods of a class to be of equal complexity, then WMC is simply the number of methods defined in each class. And the average of WMC is the average number of methods per class. [10]
2. *Depth of Inheritance tree* (DIT): This is the length of the maximum path of class hierarchy from the node to the root of the inheritance tree. [10]
3. *Number of Children of Class* (NOC): This is the number of immediate successors (subclasses) of class in a hierarchy.[10]
4. *Coupling between object classes* (CBO): An object class is coupled with another one if it invokes another one's member functions or instance variables. CBO is the number of classes to which a given class is coupled. [10]
5. *Response for Class* (RFC): This is the number of methods that can be executed in response to a message received by an object of that class. The larger the number of methods that can be invoked from a class through messages, the greater the complexity of the class. It captures the size of the response set of a class. The response set of a class is all the methods called by the local methods. RFC is the number of local methods plus the number of methods called by the local methods. [10]
6. *Lack of Cohesion on Methods* (LCOM): The cohesion of a class is indicated by how closely the local methods are related to the local instance variables in the class. High cohesion indicates good class subdivision. The LCOM metric measures the dissimilarity of methods in a class by the usage instance variables. LCOM is measured as the number of disjoint sets of local methods. Lack of cohesion increases complexity and opportunities for error during the development process. [10]

IV. CONCLUSION

Software metric are used in analyzing and maintaining the quality of the software development process and it is one of the most important process associated with SDLC. We can use the metrics to analyze various factors that impact the design and then the performance of the software product. Thus the metrics we have reviewed become an integral part of the process known as software development. The deployment of the metrics are indeed a very big task and it provides a vast array of opportunities for the programmers to refer this as a document for referencing for classifying metrics.

REFERENCES

[1] Singh Yogesh & Pradeep Bhatia, " Module Weakness—A New Measure", ACM SIGSOFT Software Engineering Notes,81,July,1998.

[2] Norman E. Fenton & Shari Lawrence Pfleeger "Software Metrics A Rigorous and Practical Approach " PWS Publishing Company, 2-1, 1997.
[3] Martin Neh, " Software Metrics for Product Assessment", McGraw Hill Book Co., UK, 2003.
[4] Henry S. & Kafura D., "Software Structure Metrics Based on Information Flow", IEEE Trans. On Software Engineering SE-7, 5, 510-518, Sept. 1981.
[5] Paul Goodman, " Practical Implementation of Software Metrics", McGraw Hill Book Co., UK, 1993.
[6] Halstead M.H., "Elements of Software Science", New York, Elsevier North Holland, 1977.
[7] Mrinal Kanti Debbarma, Swapan Debbarma, Nikhil Debbarma, Kunal Chakma, and Anupam Jamatia, "A Review and Analysis of Software Complexity Metrics inStructural Testing, March 2013.
[8] Dhawan Sunil, Wadhwa Manoj, Identification of Software Metrics for Software Projects, IJACEN, 23-27,2013.
[9] Stephen H. Kan, "Metrics and Models In Software Quality Engineering", Second Edition Pearson, 2-82 2002.
[10] Shyam R. Chidamber and Chris F. Kermer, "A Metrics Suite For Object Oriented Design", June 1994.