INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH AND TECHNOLOGY
ISSN 2582-7359
PEER REVIEW JOURNAL
IMPACT FACTOR 6.328

A STUDY OF SOFTWARE METRICS AND COMPONENT ANALYSIS

Ritu Sharma

Designation: Assistant Professor
Email: viditritu56@gmail.com

Abstract
A standard of measurement for many activities that involve some degree of measurement is software metrics. It tends to be ordered into three classes: metrics for the project, the process, and the product. One of the main reasons major software-intensive acquisition programs fail is poor size estimation. When it comes to cost, schedule, and effort, size is the most important factor. Budget overruns and late deliveries as a result of inaccurate (usually too small) predictions lower confidence and reduce support for your program. Throughout the life cycle, size estimation is a complicated process whose results must always be updated with actual counts. Size measures incorporate source lines-of-code, capability focuses, and include focuses. Size affects complexity, which has a significant impact on design errors and hidden defects, ultimately leading to quality issues, cost overruns, and schedule slips. Continuous measurement, tracking, and management of complexity are required. Another component prompting size gauge mistakes is prerequisites creep which likewise should be pattern and persistently controlled.
Software metrics are important for analyzing and improving software quality because they measure various aspects of software complexity. They provide useful information on software's external quality aspects, such as its maintainability, reusability, and reliability, according to previous research. Programming measurements give a mean of assessing the endeavors required for testing. Products and process metrics are two common categories for software metrics.
Keywords: Software metrics, Line of Code, Weight Method per Class, Response for class, Lack of Cohesion,Coupling Between object classes.

INTRODUCTION

Metrics for the process: Process measurements are known as the executives measurements and used to gauge the properties of the cycle which is utilized to get the product. Process measurements incorporate the expense measurements, endeavors measurements, headway measurements and reuse measurements. Process measurements help in foreseeing the size of conclusive framework and deciding if a venture on running as per the timetable.

Metrics for Products: The properties of the software can be measured using product metrics, which are also referred to as quality metrics. Non-reliability metrics, functionality metrics, performance metrics, usability metrics, cost metrics, size metrics, complexity metrics, and style metrics are all examples of product metrics. Items measurements help in working on the nature of various framework part and correlations between existing frameworks.

ADVANTAGE OF SOFTWARE METRICS

- in a comparative study of various software system design methodologies. for critical

study, comparison, and analysis of various programming languages in terms of their features.

- • In looking at and assessing abilities and efficiency of individuals associated with programming improvement.
- • When developing software quality specifications
- • ensuring that software system specifications and requirements are met.
- • Drawing inferences about the effort required for software system design and development.
- • in determining the code's level of complexity.
- • When making decisions about whether or not to divide the complex module further.
- • In directing the resource manager to use the resources in the right way.
- • Comparing software development costs with maintenance costs and making design compromises
- • In giving criticism to programming supervisors about the advancement and quality during different periods of programming improvement life cycle.
- • When allocating testing resources for code testing.

LIMITATIONS OF SOFTWARE METRICS

• Using software metrics isn't always simple, and in some cases, it can be difficult and expensive.

• The confirmation and support of programming measurements depends on authentic/experimental information whose legitimacy is hard to check.

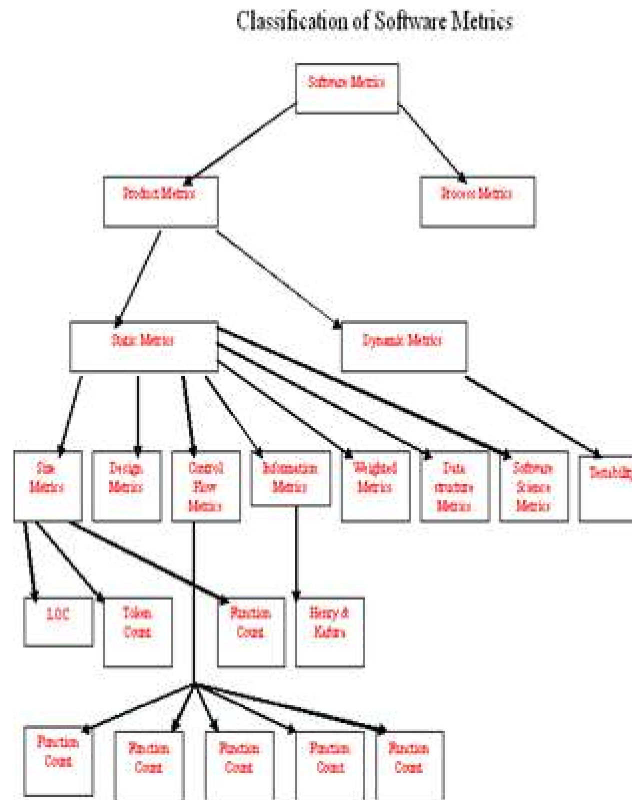
• These are useful for managing software products but not for assessing technical staff performance.

• Software metrics are typically defined and derived based on the assumption that they are not standardized and may be affected by available tools and the working environment.

• The majority of predictive models rely on estimates of particular variables that are frequently not precisely known.

• The majority of software development models are empirical and probabilistic.

CLASSIFICATION OF SOFTWARE METRICS



Size Metrics Code Line:

It is one of the earliest and easiest metrics for determining a computer program's size. It is typically used to calculate and compare programmers' productivity.

• The measure of productivity is LOC/man-month.

• Any line of program text barring remark or clear line, no matter what the quantity of assertions or portions of explanations on the line, is viewed as a Line of Code.

- Token Count
- In this measurements, a PC program is viewed as an assortment of tokens, which might be delegated either administrators or operands. These fundamental symbols can be used to define every software science metric. Tokens refer to these symbols. The fundamental metrics are n1 = the number of distinct operators.
- n2 is the number of distinct operands.
- N1 = count of all out events of administrators. N2 is the total number of operands that occur.

- The size of the program can be expressed as N = N1 + N2 Function Count, where N is the total number of tokens used.
- • The size of a huge programming item can be assessed in better manner through a bigger unit called module. A piece of code that can be compiled independently is known as a module.
- • As an illustration, suppose a software product needs n modules. It is for the most part concurred that the size of module ought to be around 50-60 line of code. Subsequently size gauge of this Product item is about n x 60 line of code.

SOFTWARE SCIENCE METRICS

Halstead's model otherwise called hypothesis of programming science, depends on the speculation that program development includes a course of mental control of the one of a kind administrators (n1) and special operands (n2). This indicates that a selection of n1 unique operators and n2 unique operands is used to construct a program with N1 operators and N2 operands. Halstead used this Model to come up with a number of programming-related equations like program level, implementation effort, language level, and so on. The fact that a program can be analyzed for a variety of characteristics, such as size, efforts, and so forth, is one important and intriguing feature of this model.

Program jargon is characterized as $n = n1 + n2$ And program real length as $N = N1 + N2$

One of the speculation of this hypothesis is that the length of a very much organized program is an element of n1 and n2 as it were. The definition of this relationship, which is referred to as the length prediction equation, is $Nh = n1 \log2 n1 + n2 \log2 n2$. Other researchers have suggested the following length estimators:

Jensen's Program Length Estimator (N1): $N1 = \text{Log2}\,(N1!) + \text{Log2}\,(n2!)$

Jensen and Vairavan tested it on Pascal-based real-time application programs and found it to be even more accurate than Halstead's estimator.

Program Length Estimator by Zipf [Nz]

$Nz = n\,[0.5772 + \ln(n)]$

where n is program vocabulary given as $n = n1 + n2$Bimlesh's Program Length Estimator [Nb]

$Nb = n1\ \text{Log2}\,(n2) + n2\ \text{Log2}\,(n1)$

where n1 : Number of unique operators which include basic operators, keywords/reserve-words and functions/procedures.

n2 : Number of unique operands.Program Volume (V)

The programming vocabulary $n = n1 + n2$ leads to anothersize measures which may be defined as :

$V = N \log 2\, n$ Potential Volume (V*)

It may be defined as $V^* = (n1^* + n2^*) \log2 (n1^* + n2^*)$ Where n1* is the minimum number of operators and n2* isthe minimum number of operands

CONTROL FLOW METRICS

The Cyclomatic Metric of McCabe: A computer program is viewed by McCabe as a collection of strongly connected directed graphs. Hubs address portions of the source code having no branches and circular segments address conceivable control stream moves during program execution.

This measurement is based on the idea of a program graph, which is used to measure and control the number of paths through a program. The topological complexity of a graph can be correlated with the complexity of a computer program.

As a measure of software complexity, McCabe proposed graph theory's cyclomatic number, V(G). The number of linearly independent paths through a program in its graph representation is equal to the cyclomatic number. The cyclomatic number, V(G), for a program control graph of type G, is as follows:

$V(G) = E - N + P$

E = The number of edges in graphs GN = The number of nodes in graphs G

P = The number of connected components in graph G.
Measurement of program complexity by Stetter: The program's control flow and data flow are both included in Stetter's metric, which can be derived from the source code. As a result, it could be viewed as a series of statements and declarations. It is provided as
P = (d1, d2, -------- , dk s1 , s2, ----- , sm)
Where d's are declarations
s's are statementsP is a program

Here, the thought of program chart has been stretch out to the idea of stream diagram. A set of edges and a set of nodes make up the program P flow graph. A hub addresses a statement or an assertion while an edge addresses one of the accompanying:

1 Control passing from one statement node, such as si, to another, such as sj.
2 Control flow from a declared statement node dj to a declared statement node si
3 Use a read access to a declared variable or constant in statement node si to move from declaration node dj to statement node si.
F(P) = E – ns + nt, where ns is the number of entry nodes and nt is the number of exit nodes, is how this measure is calculated.

INFORMATION FLOW METRICS

- • Information Flow metrics deal with complexity of this kind by tracking how information moves between system modules or components. This measurements is given by Henry and Kafura. So it is otherwise called Henry and Kafura's Measurement.
- • The measurement of the information flow between system modules serves as the foundation for this metric. Because system components are interconnected, it is sensitive to complexity. The sum of the complexities of the modules' procedures is the definition of this measure of software module complexity. A strategy contributes intricacy because of the accompanying two variables.
- • The procedure code's inherent complexity.

The intricacy because of system's associations with its current circumstance. The LOC (Lin of Code) measure has been used to account for the impact of the first factor. Henry and Kafura have defined two terms, FAN-IN and FAN-OUT, for quantifying the second factor.
A procedure's FAN-IN is the sum of the number of data structures from which it retrieves information and the number of local flows that enter that procedure.
FAN –OUT is the total number of data structures that are updated by that procedure in addition to the number of local flows generated by that procedure.
Procedure Complexity = Length * (FAN-IN * FAN- OUT)**2
Where the length is taken as LOC and the term FAN-IN
*FAN-OUT represent the total number of input –output combinations for the procedure.

NEW PROGRAM WEIGHTED COMPLEXIT MEASURE:

A program is a collection of statements that each contain an operator and an operand. As a result, a program's fundamental components are its statements, operators, and operands. The conspicuous variables which add to intricacy of a program are:
Size: The sheer volume of information required to comprehend a line program causes

problems, necessitating the use of additional resources for their upkeep. Therefore, a program's size contributes to its complexity.
Position of an Assertion: We assume that statements at the program's beginning contribute less complexity than statements at the program's deeper levels of logic because they are simpler and easier to comprehend. As a result, we create weights 1 for the initial executable statement, 2 for the second, and so on. It could be treated as positional weight (Wp).
Sort of control structure: A program's complexity is inversely proportional to the number of control structures it contains. However, we assume that distinct control structures contribute in different ways to a program's complexity. For instance, decision-making control structures like if, then, and else are less complex than iterative control structures like while, do, repeat, until, for, and to. So we dole out various loads to various control structures.
Nesting: A deeper-level statement is more difficult to comprehend and thus contributes more complexity than a simpler one. Statements at level one receive weight 1, those at level two receive weight 2, and so on to implement nesting.
The load for successive proclamations is taken as nothing. A program P's weighted Complexity measure is suggested based on these assumptions as:

Cw (P) = S (Wt)i * (m)I
I=1

OBJECT ORIENTED METRICS:

- ✓ Weight Method per Class (WMC)
- ✓ Response for Class (RFC)
- ✓ Lack of Cohesion of Methods (LCOM)
- ✓ Coupling between Object Classes (CBO)
- ✓ Depth of Inheritance Tree (DIT)
- ✓ Number of Children (NOC)

Weight Method per Class (WMC):

- Understandability, reusability, and maintainability are all measured using this metric.
- • An object can be made by using a class as a template. Classes with a lot of methods are likely to be more application-specific, which makes it harder to reuse them.
- • The set of methods demonstrates that this collection of objects has a similar structure and behavior.
- • The WMC is a count of the strategies executed inside a class or the amount of the intricacies of the techniques. However, the second measurement is more challenging to implement because inheritance prevents access to all methods in the class hierarchy.
- • The bigger the quantity of techniques in a class is the more noteworthy the effect might be on kids, since youngsters acquire each of the strategies characterized in a class.

Response for Class (RFC):

- An object sends a message to another object asking it to do something. The activity executed because of getting a message is known as a strategy.
- • The total number of methods in a set that can be called in response to a message

sent to an object is known as the RFC. All methods accessible through the class hierarchy are included in this.

- • This measurements is utilized to actually take a look at the class intricacy. The class becomes more complex when the number of methods that can be invoked via message from the class grows.

Lack of Cohesion of Methods (LCOM):

- The degree to which methods in a class are related to one another and collaborate to produce well-bounded behavior is known as cohesion.
- • The degree of similarity between methods is measured by LCOM using variables or attributes.
- • We can gauge the union for every information field in a class; work out the level of strategies that utilization the information field.
- • Divide the average percentage by 100 percent after. Lower rate shows more noteworthy information and technique union inside the class.
- • A lack of cohesion increases complexity, whereas a high cohesion indicates good class subdivision.

Coupling between Object Classes (CBO):

- • The strength of an association established by a connection between two entities is measured by coupling.
- • There are three ways that classes pair. One is that two things are said to be coupled when a message is exchanged between them. The second is that when methods declared in one class use methods or attributes of other classes, the classes are coupled. The third point is that the super class and the subclass became extremely tightly coupled as a result of inheritance.
- • The CBO is a count of how many other classes a class is coupled to. It is estimated by counting the quantity of particular non legacy related class order on which a class depends.
- • Modular design suffers from excessive coupling, which prevents reuse. Assuming the quantity of couple is bigger in programming than the aversion to changes in other in different pieces of plan.

Depth of Inheritance Tree (DIT):

- • An inheritance relationship between classes enables programmers to reuse previously defined object objects, such as variables and operators, by allowing them to do so again and again.
- • While inheritance reduces complexity by reducing the number of operations and operators, this abstraction of objects can make design and maintenance more challenging.
- • Profundity of class inside the legacy ordered progression is the greatest length from the class hub to the foundation of the tree, estimated by the quantity of predecessor classes.
- • It becomes more difficult to predict a class's behavior the higher up in the hierarchy it is, the more methods it likely has, and the more likely it is to inherit.
- • The number of methods inherited is a DIT support metric.

Number of Children (NOC):

- • In the hierarchy, the number of children is the number of immediate subclasses

that are below the class.

- • The more prominent the quantity of kids, the more noteworthy the parent reflection.
- • Because inheritance is a form of reuse, the greater the number of children, the greater the reusability.
- • If there are more students in a class, it will take more time to test the methods of that class.

CONCLUSIONS

An organization's goals-based metrics program will aid in communication, progress measurement, and eventual goal achievement. People will work hard to get what they think is important. An organization can get the information it needs to keep improving its software products, processes, and services while still focusing on the important things by using well-designed metrics with clearly stated goals. A helpful aid is a practical, systematic, and complete approach to selecting, designing, and implementing software metrics. For managing the software project, measurement has a vital role. For checking whether the project is on track, users and developers can rely on the measurement-based chart and graph. The standard set of measurements and reporting methods are especially important when the software is embedded in a product where the customers are not usually well-versed in software terminology. We investigate various software metrics utilized during software development in this paper.

References:

[1] J. K. Chhabra and V. Gupta, "A survey of dynamic software metrics," J. Comput. Sci. Technol., vol. 25, pp. 1016-1029, 2010.

[2] O. Gómez, H. Oktaba, M. Piattini, and F. García, "A Systematic Review Measurement in Software Engineering: State-of-the-Art in Measures," in Communications in Computer and Information Science- Software and Data Technologies. vol. 10, J. Filipe, et al., Eds., ed: Springer Berlin Heidelberg, 2008, pp. 165-176.

[3] B. Kitchenham, "What's up with software metrics? - A preliminary mapping study," Journal of Systems and Software, vol. 83, pp. 37-51, 2010.

[4] B. Cornelissen, A. Zaidman, A. v. Deursen, L. Moonen, and R. Koschke, "A Systematic Survey of Program Comprehension through Dynamic Analysis," IEEE Transactions on Software Engineering, vol. 35, pp. 684-702, 2009.

[5] Chidamber, Shyam and Kemerer, Chris, “A metrics Suite for Object Oriented Design”, IEEE Transactions on Software Engineering, June, 1994, pp. 476-492.

[6] Lorenz, Mark and Kidd, Jeff, Object Oriented Software metrics, Prentice Hall Publishing, 1994.

[7] Victor R. Basili, Lionel Briand and Walcelio L. Melo “A validation of object-oriented design metrics as quality indicators” Technical report, Uni. of Maryland, Deptt. of computer science, MD, USA.April 1995.

[8] Rajender Singh, Grover P.S., “A new program weighted complexity metrics” proc. International conference on Software Engg. (CONSEG’97), January Chennai (Madras) India, pp 33-39

I. Brooks “Object oriented metrics collection and evaluation with software process” presented atOOPSLA’93 Workshop on Processes and Metrics for Object Oriented Software development, Washington, DC.

[9] “Software quality metrics for Object Oriented System Environments” by Software Assurance Technology Center,National Aeronautics and Space Administration.