ISSN (Online): 2230-7893
www.IJCEM.org

A Study of Software Metrics

Gurdev Singh[1], Dilbag Singh[2], Vikram Singh[3]

[1] Assistant Professor, JIET Jind. gurujangra@gmail.com

[2] Professor, Dept. of CSE, Ch. Devi Lal University Sirsa

[3] Professor, Dept. of CSE, Ch. Devi Lal University Sirsa

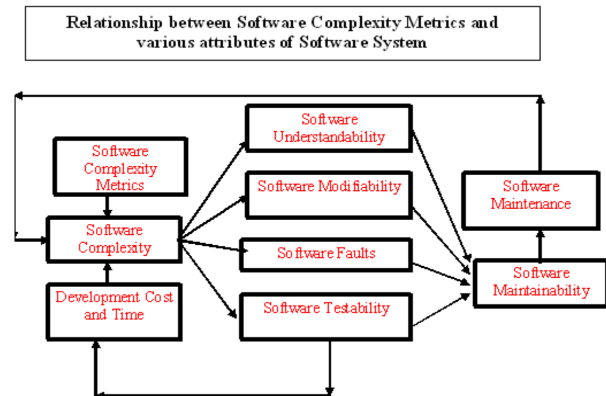
Abstract

Poor size estimation is one of the main reasons major software-intensive acquisition programs ultimately fail. Size is the critical factor in determining cost, schedule, and effort. The failure to accurately predict (usually too small) results in budget overruns and late deliveries which undermine confidence and erode support for your program. Size estimation is a complicated activity, the results of which must be constantly updated with actual counts throughout the life cycle. Size measures include source lines-of-code, function points, and feature points. Complexity is a function of size, which greatly impacts design errors and latent defects, ultimately resulting in quality problems, cost overruns, and schedule slips. Complexity must be continuously measured, tracked, and controlled. Another factor leading to size estimate inaccuracies is requirements creep which also must be baseline and diligently controlled.

Software metrics measure different aspects of software complexity and therefore play an important role in analyzing and improving software quality. Pervious research has indicated that they provide useful information on external quality aspects of software such as its maintainability, reusability and reliability. Software metrics provide a mean of estimating the efforts needed for testing. Software metrics are often categorized into products and process metrics.

Keywords: LOC-Line of Code, WMC-Weight Method per Class, RFC-Response for class, LCOM- Lack of Cohesion, CBO- Coupling Between object classes, DIT- Depth of Inheritance Tree.

1. RELATIONSHIP BETWEEN SOFTWARE COMPLEXITY METRICS AND VARIOUS ATTRIBUTES OF SOFTWARE SYSTEM:



2. TYPE OF SOFTWARE METRICS:

2.1 Process Metrics: Process metrics are known as management metrics and used to measure the properties of the process which is used to obtain the software. Process metrics include the cost metrics, efforts metrics, advancement metrics and reuse metrics. Process metrics help in predicting the size of final system & determining whether a project on running according to the schedule.

2.2 Products Metrics: Product metrics are also known as quality metrics and is used to measure the properties of the software. Product metrics includes product non reliability metrics, functionality metrics, performance metrics, usability metrics, cost metrics, size metrics, complexity metrics and style metrics. Products metrics help in improving the quality of different system component & comparisons between existing systems.

3. ADVANTAGE OF SOFTWARE METRICS:

- In Comparative study of various design methodology of software systems.
- For analysis, comparison and critical study of various programming language with respect to their characteristics.

ISSN (Online): 2230-7893
www.IJCEM.org

- In comparing and evaluating capabilities and productivity of people involved in software development.
- In the preparation of software quality specifications.
- In the verification of compliance of software systems requirements and specifications.

- In making inference about the effort to be put in the design and development of the software systems.

- In getting an idea about the complexity of the code.

- In taking decisions regarding further division of complex module is to be done or not.

- In providing guidance to resource manager for their proper utilization.

- In comparison and making design tradeoffs between software development and maintenance cost.

- In providing feedback to software managers about the progress and quality during various phases of software development life cycle.

- In allocation of testing resources for testing the code.

4. LIMITATION OF SOFTWARE METRICS:

- The application of software metrics is not always easy and in some cases it is difficult and costly.

- The verification and justification of software metrics is based on historical/empirical data whose validity is difficult to verify.

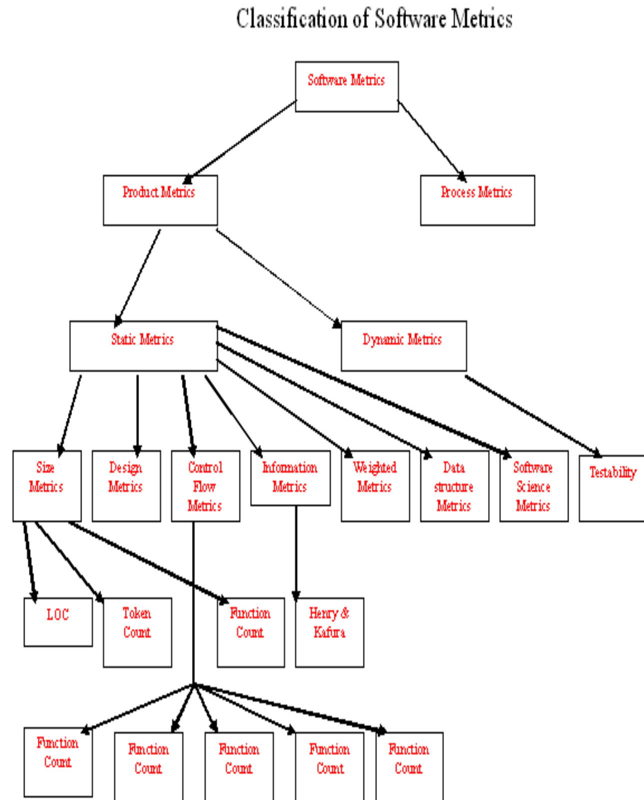
- These are useful for managing the software products but not for evaluating performance of the technical staff.

- The definition and derivation of Software metrics is generally based on assuming which are not standardized and may depend upon tools available and working environment.

- Most of the predictive models rely on estimates of certain variables which are often not known exactly.

- Most of the software development models are probabilistic and empirical.

5. CLASSIFICATION OF SOFTWARE METRICS:



6. SIZE METRICS:

6.1 Line of Code: It is one of the earliest and simpler metrics for calculating the size of computer program. It is generally used in calculating and comparing the productivity of programmers.

- Productivity is measured as LOC/man-month.
- Any line of program text excluding comment or blank line, regardless of the number of statements or parts of statements on the line, is considered a Line of Code.

6.2 Token Count:

In this metrics, a computer program is considered to be a collection of tokens, which may be classified as either operators or operands. All software science metrics can be defined in terms of these basic symbols. These symbols are called as token. The basic measures are

$n1$ = count of unique operators.

ISSN (Online): 2230-7893
www.IJCEM.org

$n2$ = count of unique operands.
$N1$ = count of total occurrences of operators.
$N2$ = count of total occurrence of operands.

In terms of the total tokens used, the size of the program can be expressed as $N = N1 + N2$

6.3 Function Count:

- The size of a large software product can be estimated in better way through a larger unit called module. A module can be defined as segment of code which may be compiled independently.
- For example, let a software product require n modules. It is generally agreed that the size of module should be about 50-60 line of code. Therefore size estimate of this Software product is about n x 60 line of code.

7. SOFTWARE SCIENCE METRICS:

Halstead's model also known as theory of software science, is based on the hypothesis that program construction involves a process of mental manipulation of the unique operators (n1) and unique operands (n2). It means that a program of N1 operators and N2 operands is constructed by selecting from n1 unique operators and n2 unique operands. By using this Model, Halstead derived a number of equations related to programming such as program level, the implementation effort, language level and so on. An important and interesting characteristics if this model is that a program can be analyzed for various feature like size, efforts etc.
Program vocabulary is defined as $n = n1 + n2$
And program actual length as $N = N1 + N2$
One of the hypothesis of this theory is that the length of a well-structured program is a function of n1 and n2 only. This relationship is known as length prediction equation and is defined as
$Nh = n1 \log2\ n1 + n2 \log2\ n2$
The following length estimators have been suggested by some other researchers:

Jensen's Program Length Estimator [N1]

It is described as
$N1 = \text{Log2} (N1!) + \text{Log2} (n2!)$
It was applied and validated by Jensen and Vairavan for real time application programs written in Pascal and found even more accurate results than Halstead's estimator.

Zipf's Program Length Estimator [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 another size 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* is the minimum number of operands.

8. CONTROL FLOW METRICS:

8.1 *McCabe's Cyclomatic Metric:* McCabe interprets a computer program as a set of strongly connected directed graph. Nodes represent parts of the source code having no branches and arcs represent possible control flow transfers during program execution.
The notion of program graph has been used for this measure and it is used to measure and control the number of paths through a program. The complexity of a computer program Can be correlated with the topological complexity of a graph.
McCabe proposed the cyclomatic number, V(G) of graph theory as an indicator of software complexity. The cyclomatic number is equal to the number of linearly independent paths through a program in its graphs representation. For a program control graph G, cyclomatic number , V(G), is given as:
$V(G) = E - N + P$
E = The number of edges in graphs G
N = The number of nodes in graphs G
P = The number of connected components in graph G.

8.2 *Stetter's Program Complexity Measure:* Stetter's metric accounts for the data flow along with thecontrol flow of the program which can be calculated from the source code. So it may be view as a sequence of declaration and statements. It is given as
P = (d1, d2, -------- , dk s1 , s2, ---------------, sm)
Where d's are declarations
s's are statements
P is a program
Here, the notion of program graph has been extend to the notion of flow graph. A flow graph of a program P can be

defined as a set of nodes and a set of edges. A node represents a declaration or a statement while an edge represents one of the following:

1 Flow of control from one statement node say si to another sj.
2 Control flow from a statement node dj to a statement node si which is declared in dj.
3 Flow from a declaration node dj to statement node si through a read access of a variable or a constant in si which is declared in dj.
This measure is defined as F(P) = E – ns + nt
Where ns = number of entry nodes
nt = number of exit nodes

9. INFORMATION FLOW METRICS:

- Information Flow metrics deal with this type of complexity by observing the flow of information among system components or modules.This metrics is given by Henry and Kafura. So it is also known as Henry and Kafura's Metric.

- This metrics is based on the measurement of the information flow among system modules. It is sensitive to the complexity due to interconnection among system component. This measure includes complexity of a software module is defined to be the sum of complexities of the procedures included in the module. A procedure contributes complexity due to the following two factors.

1. The complexity of the procedure code itself.
2. The complexity due to procedure's connections to its environment. The effect of the first factor has been included through LOC (Lin Of Code) measure. For the quantification of second factor, Henry and Kafura have defined wo terms, namely FAN-IN and FAN-OUT.

FAN-IN of a procedure is the number of local flows into that procedure plus the number of data structures from which this procedure retrieve information.
FAN –OUT is the number of local flows from that procedure plus the number of data structures which that procedure updates.
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.

10. NEW PROGRAM WEIGHTED COMPLEXIT MEASURE:

A program is a set of statements which in turn include operators and operands. Thus the program statements, operators and operands are basic units of a program. The prominent factors which contribute to complexity of a program are:

10.1 Size: Line program incur problem just by virtue of volume of the Information that must be absorbed to understand the program and more resources have to be used in their maintenance. Therefore size is a factor which adds complexity to a program.

10.2 Position of a Statement: We assume that the statements which are at the beginning of the program logic are simple and hence easily understandable and thus contribute less complexity than those which are at deeper level of the logic of a program. So we design a weight 1 to first executable statement and 2 to second and so on. It may be treated as positional weight (Wp).

10.3 Type of control structure: A program with more control structures is considered to be more complex and vice versa. But, we assume that different control structures contribute to the complexity of a program differently. For example, iterative control structures like while..do, repeat … until, for .. to .. do contribute more complexity than decision making control structure like if.. then.. Else. So we assign different weights to different control structures

10.4 Nesting: A statement which is at deeper level is harder to understand and thus contribute more complexity than otherwise. We take effect of nesting by assigning weight 1 to statements at level one, Weight 2 for those statements which are at level 2 and so on.
The weight for sequential statements is taken as zero.By taking these assumptions into account, a weighted Complexity measure of a program P is suggested as:

$$Cw(P) = \sum_{I=1}^{j} (Wt)i * (m)I$$

11. 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)

11.1 Weight Method per Class (WMC):

This metric is used to measure the understandability, reusability and maintainability.

- A class is a template from which objects can be created. Classes with large number of methods are likely to more application specific, limiting the possibility of reuse.
- This set of objects shares a common structure and a common behavior manifested by the set of methods.
- The WMC is a count of the methods implemented within a class or the sum of the complexities of the methods. But the second measurement is more difficult to implement because not all methods are accessible within the class hierarchy because of inheritance.
- The larger the number of methods in a class is the greater the impact may be on children, since children inherit all of the methods defined in a class.

11.2 Response for Class (RFC):

A message is a request that an object makes to another object to perform an operation. The operation executed as a result of receiving a message is called a method.

- The RFC is the total number of all methods within a set that can be invoked in response to message sent to an object. This includes all methods accessible within the class hierarchy.
- This metrics is used to check the class complexity. If the number of method is larger that can be invoked from class through message than the complexity of the class is increase**.**

11.3 Lack of Cohesion of Methods (LCOM):

Cohesion is the degree to which methods within a class are related to one another and work together to provide well bounded behavior.

- LCOM uses variable or attributes to measure the degree of similarity between methods.
- We can measure the cohesion for each data field in a class; calculate the percentage of methods that use the data field.
- Average the percentage, then subtract from 100 percent. Lower percentage indicates greater data and method cohesion within the class.
- High cohesion indicates good class subdivision while a lack of cohesion increases the complexity.

11.4 Coupling between Object Classes (CBO):

- Coupling is a measure of strength of association established by a connection from one entity to another.
- Classes are couple in three ways. One is, when a message is passed between objects, the object are said to be coupled. Second one is, the classes are coupled when methods declared in one class use methods or attributes of the other classes. Third on is, inheritance introduced significant tight coupling between super class and subclass.
- CBO is a count of the number of other classes to which a class is coupled. It is measured by counting the number of distinct non inheritance related class hierarchy on which a class depends.
- Excessive coupling is detrimental to modular design and prevent reuse. If the number of couple is larger in software than the sensitivity to changes in other in other parts of design.

11.5 Depth of Inheritance Tree (DIT):

- Inheritance is a type of relationship among classes that enables programmers to reuse previously defined object objects, including variables & operators.

- Inheritance decrease the complexity by reducing the number of operations and operators, but this abstraction of objects can make maintenance and design more difficult.
- Depth of class within the inheritance hierarchy is the maximum length from the class node to the root of the tree, measured by the number of ancestor classes.
- The deeper a class within the hierarchy, the greater the number of methods and is likely to inherit, making it more complex to predict its behavior.
- A support metric for DIT is the number of methods inherited.

11.6 Number of Children (NOC):

- The number of children is the number of immediate subclasses subordinates to class in the hierarchy.
- The greater the number of children, the greater the parent abstraction.
- The greater the number of children, greater the reusability, since the inheritance is a form of reuse.
- If the number of children in class is larger than it require more testing time for testing the methods of that class.

12 CONCLUSIONS:

A metrics program that is based on the goals of an organization will help communicate, measure progress towards, and eventually attain those goals. People will work to accomplish what they believe to be important. Well-designed metrics with documented objectives can help an organization obtain the information it needs to continue to improve its software products, processes, and services while maintaining a focus on what is important. A practical, systematic, start-to-finish method of selecting, designing, and implementing software metrics is a valuable aid. In this paper we study different type of software metrics which are used during the software development.

References:

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

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

[3] 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.

[4] "The Role of Object Oriented metrics" from archive.eiffel.com/ doc/ manuals/ technology.

[5] 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

[6] I. Brooks "Object oriented metrics collection and evaluation with software process" presented at OOPSLA'93 Workshop on Processes and Metrics for Object Oriented Software development, Washington, DC.

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