SOFTWARE ENGINEERING

Unit 2

Software Metrics:

A metric is a measurement of the level at which any impute belongs to a system product or process.

Software metrics will be useful only if they are characterized effectively and validated so that their worth is proven. There are 4 functions related to software metrics:

1. Planning
2. Organizing
3. Controlling
4. Improving

Characteristics of software Metrics:

1. **Quantitative:** Metrics must possess quantitative nature. It means metrics can be expressed in values.
2. **Understandable:** Metric computation should be easily understood, and the method of computing metrics should be clearly defined.
3. **Applicability:** Metrics should be applicable in the initial phases of the development of the software.
4. **Repeatable:** The metric values should be the same when measured repeatedly and consistent in nature.
5. **Economical:** The computation of metrics should be economical.
6. **Language Independent:** Metrics should not depend on any programming language.

Classification of Software Metrics:

There are 3 types of software metrics:

1. **Product Metrics:** Product metrics are used to evaluate the state of the product, tracing risks and undercover prospective problem areas. The ability of the team to control quality is evaluated. Examples include lines of code, cyclomatic complexity, code coverage, defect density, and code maintainability index.
2. **Process Metrics:** Process metrics pay particular attention to enhancing the long-term process of the team or organization. Examples include effort variance, schedule variance, defect injection rate, and lead time.
3. **Project Metrics:** The project matrix describes the project characteristic and execution process. Examples include effort estimation accuracy, schedule deviation, cost variance, and productivity.
    - Number of software developer
    - Staffing patterns over the life cycle of software
    - Cost and schedule
    - Productivity

Advantages of Software Metrics :

1. Reduction in cost or budget.
2. It helps to identify the particular area for improvising.
3. It helps to increase the product quality.
4. Managing the workloads and teams.
5. Reduction in overall time to produce the product,.
6. It helps to determine the complexity of the code and to test the code with resources.
7. It helps in providing effective planning, controlling and managing of the entire product.

Disadvantages of Software Metrics :

1. It is expensive and difficult to implement the metrics in some cases.
2. Performance of the entire team or an individual from the team can't be determined. Only the performance of the product is determined.
3. Sometimes the quality of the product is not met with the expectation.
4. It leads to measure the unwanted data which is wastage of time.
5. Measuring the incorrect data leads to make wrong decision making.

Size Oriented Metrics

LOC Metrics

It is one of the earliest and simpler metrics for calculating the size of the computer program. It is generally used in calculating and comparing the productivity of programmers. These metrics are derived by normalizing the quality and productivity measures by considering the size of the product as a metric.

Following are the points regarding LOC measures:

1. In size-oriented metrics, LOC is considered to be the normalization value.
2. It is an older method that was developed when FORTRAN and COBOL programming were very popular.
3. Productivity is defined as KLOC / EFFORT, where effort is measured in person-months.
4. Size-oriented metrics depend on the programming language used.
5. As productivity depends on KLOC, so assembly language code will have more productivity.
6. LOC measure requires a level of detail which may not be practically achievable.
7. The more expressive is the programming language, the lower is the productivity.
8. LOC method of measurement does not apply to projects that deal with visual (GUI-based) programming. As already explained, Graphical User Interfaces (GUIs) use forms basically. LOC metric is not applicable here.
9. It requires that all organizations must use the same method for counting LOC. This is so because some organizations use only executable statements, some useful comments, and some do not. Thus, the standard needs to be established.
10. These metrics are not universally accepted.

Based on the LOC/KLOC count of software, many other metrics can be computed:

a. Errors/KLOC.
b. $/ KLOC.
c. Defects/KLOC.
d. Pages of documentation/KLOC.
e. Errors/PM.
f. Productivity = KLOC/PM (effort is measured in person-months).
g. $/ Page of documentation.

Advantages of LOC

1. Simple to measure

Disadvantage of LOC

1. It is defined on the code. For example, it cannot measure the size of the specification.
2. It characterizes only one specific view of size, namely length, it takes no account of functionality or complexity
3. Bad software design may cause an excessive line of code
4. It is language dependent
5. Users cannot easily understand it

Token Count

In these 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 a token.

The basic measures are

n1 =count of unique operators.
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.

Halstead metrics are:

Program Volume (V)

The unit of measurement of volume is the standard unit for size "bits." It is the actual size of a program if a uniform binary encoding for the vocabulary is used.

$V=N*\log_2 n$

Program Level (L)

The value of L ranges between zero and one, with L=1 representing a program written at the highest possible level (i.e., with minimum size).

$L=V^*/V$

Program Difficulty

The difficulty level or error-proneness (D) of the program is proportional to the number of the unique operator in the program.

$D= (n1/2) * (N2/n2)$

Programming Effort (E)

The unit of measurement of E is elementary mental discriminations.

$E=V/L=D*V$

Estimated Program Length

According to Halstead, The first Hypothesis of software science is that the length of a well-structured program is a function only of the number of unique operators and operands.

$N=N1+N2$

And estimated program length is denoted by N^

$$N\hat{} = n1\log_2 n1 + n2\log_2 n2$$

The following alternate expressions have been published to estimate program length:

- $N_J = \log_2 (n1!) + \log_2 (n2!)$
- $N_B = n1 * \log_2 n2 + n2 * \log2n1$
- $N_C = n1 * sqrt(n1) + n2 * sqrt(n2)$
- $N_S = (n * \log_2 n) / 2$

Potential Minimum Volume

The potential minimum volume V* is defined as the volume of the most short program in which a problem can be coded.

$$V^* = (2 + n2^*) * \log_2 (2 + n2^*)$$

Here, n2* is the count of unique input and output parameters

Size of Vocabulary (n)

The size of the vocabulary of a program, which consists of the number of unique tokens used to build a program, is defined as:

$$n=n1+n2$$

where

n=vocabulary of a program
n1=number of unique operators
n2=number of unique operands

Language Level - Shows the algorithm implementation program language level. The same algorithm demands additional effort if it is written in a low-level program language. For example, it is easier to program in Pascal than in Assembler.

L' = V / D / D
lambda = L * V* = L^2 * V

Language levels

Language	Language level λ	Variance σ
PL/1	1.53	0.92
ALGOL	1.21	0.74
FORTRAN	1.14	0.81
CDC Assembly	0.88	0.42
PASCAL	2.54	-
APL	2.42	-
C	0.857	0.445

Counting rules for C language

1. Comments are not considered.
2. The identifier and function declarations are not considered
3. All the variables and constants are considered operands.
4. Global variables used in different modules of the same program are counted as multiple occurrences of the same variable.
5. Local variables with the same name in different functions are counted as unique operands.
6. Functions calls are considered as operators.
7. All looping statements e.g., do {...} while ( ), while ( ) {...}, for ( ) {...}, all control statements e.g., if ( ) {...}, if ( ) {...} else {...}, etc. are considered as operators.
8. In control construct switch ( ) {case:...}, switch as well as all the case statements are considered as operators.
9. The reserve words like return, default, continue, break, sizeof, etc., are considered as operators.
10. All the brackets, commas, and terminators are considered as operators.

11. GOTO is counted as an operator, and the label is counted as an operand.
12. The unary and binary occurrence of "+" and "-" are dealt with separately. Similarly "*" (multiplication operator) are dealt separately.
13. In the array variables such as "array-name [index]" "array-name" and "index" are considered as operands and [ ] is considered an operator.
14. In the structure variables such as "struct-name, member-name" or "struct-name -> member-name," struct-name, member-name are considered as operands and '.', '->' are taken as operators. Some names of member elements in different structure variables are counted as unique operands.
15. All the hash directive is ignored.

Example: Consider the sorting program as shown in fig: List out the operators and operands and also calculate the value of software science measure like n, N, V, E, λ ,etc.

Solution: The list of operators and operands is given in the table

Operators	Occurrences	Operands	Occurrences
Int	4	SORT	1
()	5	x	7
,	4	n	3
[]	7	i	8
If	2	j	7
<	2	save	3
;	11	im1	3
For	2	2	2

=	6	1	3
-	1	0	1
<=	2	-	-
++	2	-	-
Return	2	-	-
{}	3	-	-
n1=14	N1=53	n2=10	N2=38

Here N1=53 and N2=38. The program length N=N1+N2=53+38=91

Vocabulary of the program n=n1+n2=14+10=24

Volume V= N * $\log_2$N=91 x $\log_2$ 24=417 bits.

The estimate program length N of the program

= 14 $\log_2$14+10 $\log_2$)10
= 14 * 3.81+10 * 3.32
= 53.34+33.2=86.45

Conceptually unique input and output parameters are represented by n2*.

n2*=3 {x: array holding the integer to be sorted. This is used as both input and output}

{N: the size of the array to be sorted}

The Potential Volume V*=5$\log_2$5=11.6

Since L=V*/V

$$= \frac{11.6}{417} = 0.027$$

D=I/L

$$= \frac{1}{0.027} = 37.03$$

Estimated Program Level

$$L\hat{} = \frac{2}{n1} X \frac{n2}{N2} = \frac{2}{14} X \frac{10}{38} = 0.038$$

We may use another formula

V^=V x L^= 417 x 0.038=15.67

E^=V/L^=D^ x V

$$= \frac{417}{0.038} = 10973.68$$

Therefore, 10974 elementary mental discrimination is required to construct the program.

$$T = \frac{E}{\beta} = \frac{10974}{18} = 610 \text{ seconds} = 10 \text{ minutes}$$

This is probably a reasonable time to produce the program, which is very simple.

Data Structure Metrics

Essentially the need for software development and other activities are to process data. Some data is input to a system, program or module; some data may be used internally, and some data is the output from a system, program, or module.

Example:

Program	Data Input	Internal Data	Data Output

Payroll	Name/Social Security No./Pay rate/Number of hours worked	Withholding rates Overtime Factors Insurance Premium Rates	Gross Pay withholding Net Pay Pay Ledgers
Spreadsheet	Item Names/Item Amounts/Relationships among Items	Cell computations Subtotal	Spreadsheet of items and totals
Software Planner	Program Size/No of Software developer on team	Model Parameter Constants Coefficients	Est. project effort Est. project duration

That's why an important set of metrics which capture in the amount of data input, processed in an output form software. A count of this data structure is called Data Structured Metrics. In these concentrations is on variables (and given constant) within each module & ignores the input-output dependencies.

There are some Data Structure metrics to compute the effort and time required to complete the project. There metrics are:

1. The Amount of Data.
2. The Usage of data within a Module.
3. Program weakness.
4. The sharing of Data among Modules.

1. The Amount of Data: To measure the amount of Data, there are further many different metrics, and these are:

- **Number of variable (VARS):** In this metric, the Number of variables used in the program is counted.
- **Number of Operands (η_2):** In this metric, the Number of operands used in the program is counted.
  η_2 = VARS + Constants + Labels
- **Total number of occurrence of the variable (N2):** In this metric, the total number of occurrence of the variables are computed

2. The Usage of data within a Module: The measure this metric, the average numbers of live variables are computed. A variable is live from its first to its last references within the procedure.

$$\text{Average no of Live variables (LV)} = \frac{\text{Sum of count live variables}}{\text{Sum of count of executable statements}}$$

For Example: If we want to characterize the average number of live variables for a program having modules, we can use this equation.

$$\overline{\text{LV}}\text{ program} = \frac{\sum_{i=1}^{m} \overline{\text{LV}}_i}{m}$$

Where ($\boxed{\text{LV}}$) is the average live variable metric computed from the ith module. This equation could compute the average span size ($\boxed{\text{SP}}$) for a program of n spans.

$$(\overline{\text{SP}})\text{ program} = \frac{\sum_{i=1}^{n} \text{SP}_i}{n}$$

3. Program weakness: Program weakness depends on its Modules weakness. If Modules are weak(less Cohesive), then it increases the effort and time metrics required to complete the project.

$$\text{Average life of variables } (\gamma) = \frac{\text{Sum of count live variables}}{\text{Sum of count of executable statements}}$$

Module Weakness (WM) = $\boxed{\text{LV}}$* γ

A program is normally a combination of various modules; hence, program weakness can be a useful measure and is defined as:

$$\mathbf{WP} = \frac{\left(\sum_{i=1}^{m} \text{WM}_i\right)}{m}$$

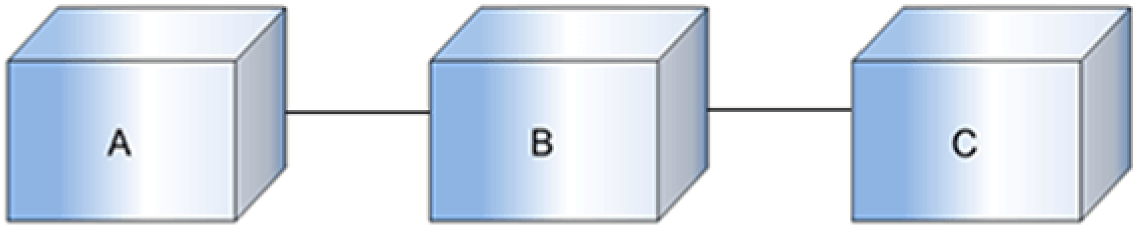
Where

WM$_i$: Weakness of the ith module

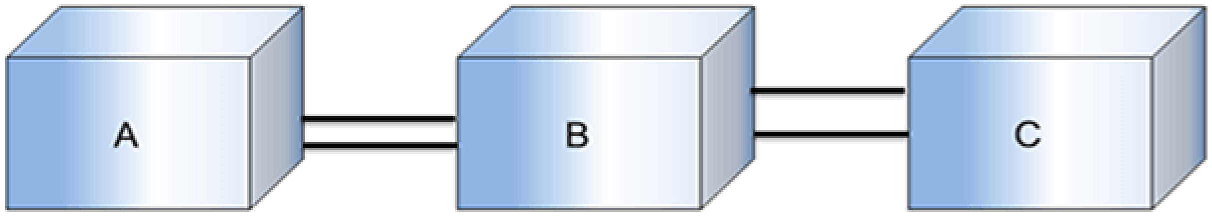
WP: Weakness of the program

m: No of modules in the program

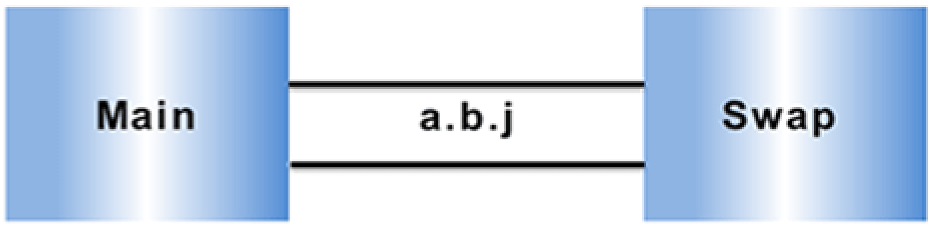
4.There Sharing of Data among Module: As the data sharing between the Modules increases (higher Coupling), no parameter passing between Modules also increased, As a result, more effort and time are required to complete the project. So Sharing Data among Module is an important metrics to calculate effort and time.



Three modules from an imaginary program



"Pipes"of data shared among the modules



The data shared in program bubble

Information Flow Metrics

The other set of metrics we would live to consider are known as Information Flow Metrics. The basis of information flow metrics is found upon the following concept the simplest system consists of the component, and it is the work that these components do and how they are fitted together that identify the complexity of the system. The following are the working definitions that are used in Information flow:

Component: Any element identified by decomposing a (software) system into it's constituent's parts.

Cohesion: The degree to which a component performs a single function.

Coupling: The term used to describe the degree of linkage between one component to others in the same system.

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 the complexity of a software module is defined to be the sum of complexities of the procedures included in the module. A process contributes complexity due to the following two factors.

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

FAN-IN: 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: FAN-OUT is the number of local flows from that procedure plus the number of data structures which that procedure updates.