Software Metrics

A software metric is a measure of software characteristics which are measurable or countable. Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.

Within the software development process, many metrics are that are all connected. Software metrics are similar to the four functions of management: Planning, Organization, Control, or Improvement.

Classification of Software Metrics

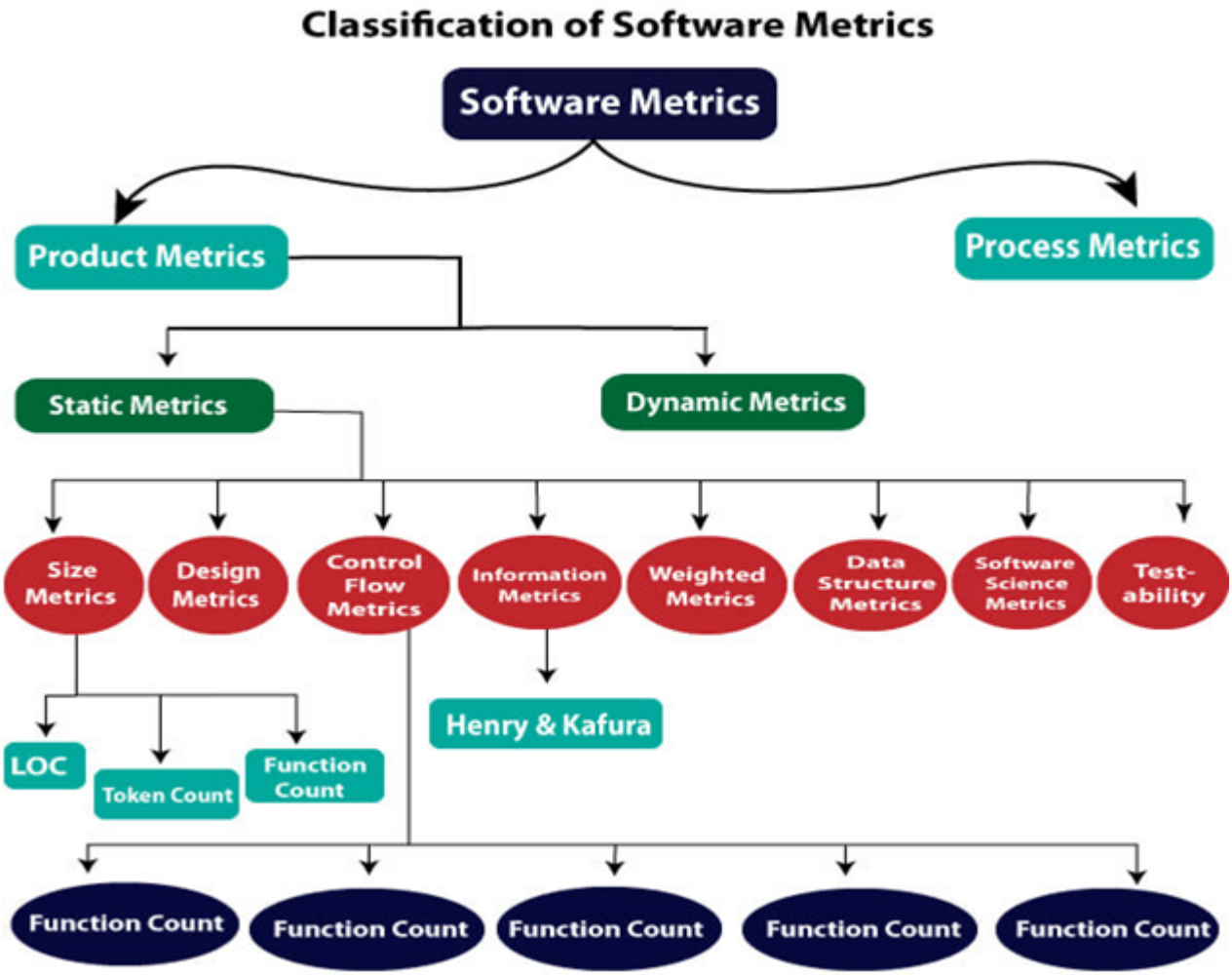
Software metrics can be classified into two types as follows:

1. Product Metrics: These are the measures of various characteristics of the software product. The two important software characteristics are:

1. Size and complexity of software.
2. Quality and reliability of software.

These metrics can be computed for different stages of SDLC.

2. Process Metrics: These are the measures of various characteristics of the software development process. For example, the efficiency of fault detection. They are used to measure the characteristics of methods, techniques, and tools that are used for developing software.



Types of Metrics

Internal metrics: Internal metrics are the metrics used for measuring properties that are viewed to be of greater importance to a software developer. For example, Lines of Code (LOC) measure.

External metrics: External metrics are the metrics used for measuring properties that are viewed to be of greater importance to the user, e.g., portability, reliability, functionality, usability, etc.

Hybrid metrics: Hybrid metrics are the metrics that combine product, process, and resource metrics. For example, cost per FP where FP stands for Function Point Metric.

Project metrics: Project metrics are the metrics used by the project manager to check the project's progress. Data from the past projects are used to collect various metrics, like time and cost; these estimates are used as a base of new software. Note that as the project proceeds, the project manager will check its progress from time-to-time and will compare the effort, cost, and time with the original effort, cost and time. Also

understand that these metrics are used to decrease the development costs, time efforts and risks. The project quality can also be improved. As quality improves, the number of errors and time, as well as cost required, is also reduced.

Advantage of Software Metrics

Comparative study of various design methodology of software systems.

For analysis, comparison, and critical study of different programming language concerning their characteristics.

In comparing and evaluating the 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 a complex module is to be done or not.

In guiding 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 the software development life cycle.

In the allocation of testing resources for testing the code.

Disadvantage 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 are based on historical/empirical data whose validity is difficult to verify.

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

The definition and derivation of Software metrics are usually 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 precisely.

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

Halstead's Software Metrics

According to Halstead's "A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operand."

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^

$$N^ = n1\log_2 n1 + n2\log_2 n2$$

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

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

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.

Functional Point (FP) Analysis

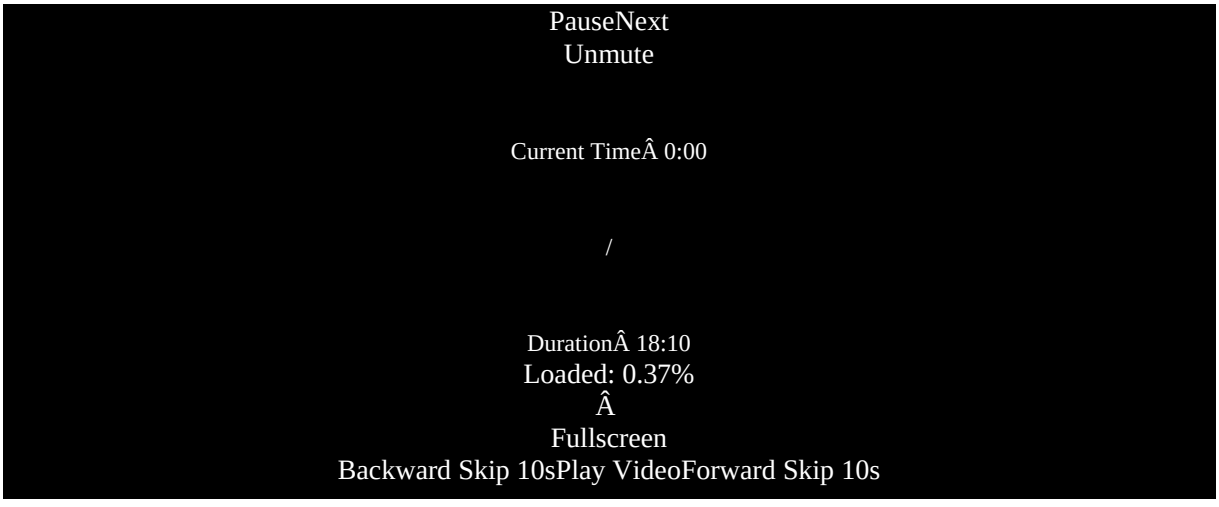
Allan J. Albrecht initially developed function Point Analysis in 1979 at IBM and it has been further modified by the International Function Point Users Group (IFPUG). FPA is used to make estimate of the software project, including its testing in terms of functionality or function size of the software product. However, functional point analysis may be used for the test estimation of the product. The functional size of the product is measured in terms of the function point, which is a standard of measurement to measure the software application.

Objectives of FPA

The basic and primary purpose of the functional point analysis is to measure and provide the software application functional size to the client, customer, and the stakeholder on their request. Further, it is used to measure the software project development along with its maintenance, consistently throughout the project irrespective of the tools and the technologies.

Following are the points regarding FPs

1. FPs of an application is found out by counting the number and types of functions used in the applications. Various functions used in an application can be put under five types, as shown in Table:



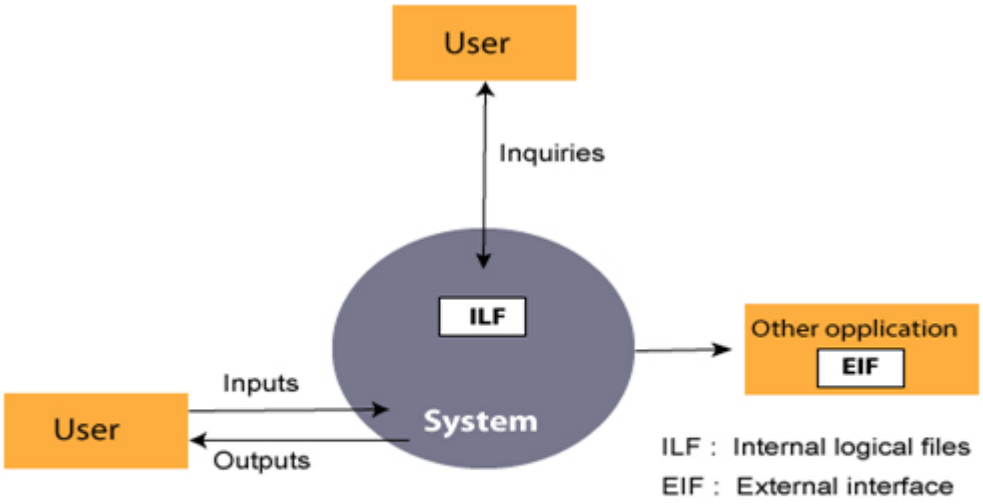
Types of FP Attributes

Measurements Parameters	Examples

1.Number of External Inputs(EI)	Input screen and tables
2. Number of External Output (EO)	Output screens and reports
3. Number of external inquiries (EQ)	Prompts and interrupts.
4. Number of internal files (ILF)	Databases and directories
5. Number of external interfaces (EIF)	Shared databases and shared routines.

All these parameters are then individually assessed for complexity.

The FPA functional units are shown in Fig:



FPAs Functional Units System

2. FP characterizes the complexity of the software system and hence can be used to depict the project time and the manpower requirement.

3. The effort required to develop the project depends on what the software does.

4. FP is programming language independent.

5. FP method is used for data processing systems, business systems like information systems.

6. The five parameters mentioned above are also known as information domain characteristics.

7. All the parameters mentioned above are assigned some weights that have been experimentally determined and are shown in Table

Weights of 5-FP Attributes

Measurement Parameter	Low	Average	High
1. Number of external inputs (EI)	7	10	15
2. Number of external outputs (EO)	5	7	10
3. Number of external inquiries (EQ)	3	4	6
4. Number of internal files (ILF)	4	5	7
5. Number of external interfaces (EIF)	3	4	6

The functional complexities are multiplied with the corresponding weights against each function, and the values are added up to determine the UFP (Unadjusted Function Point) of the subsystem.

Computing FPs

Measurement Parameter	Count		Weighing factor			
			Simple	Average	Complex	
1. Number of external inputs (EI)	—	*	3	4	6 =	—
2. Number of external Output (EO)	—	*	4	5	7 =	—
3. Number of external Inquiries (EQ)	—	*	3	4	6 =	—
4. Number of internal Files (ILF)	—	*	7	10	15 =	—
5. Number of external interfaces(EIF)	—	*	5	7	10 =	—
Count-total →						

Here that weighing factor will be simple, average, or complex for a measurement parameter type.

The Function Point (FP) is thus calculated with the following formula.

$$\mathbf{FP = Count\text{-}total * [0.65 + 0.01 * \sum(f_i)] = Count\text{-}total * CAF}$$

where Count-total is obtained from the above Table.

$$\mathbf{CAF = [0.65 + 0.01 * \sum(f_i)]}$$

and $\mathbf{\sum(f_i)}$ is the sum of all 14 questionnaires and show the complexity adjustment value/ factor-CAF (where i ranges from 1 to 14). Usually, a student is provided with the value of $\sum(f_i)$

Also note that $\mathbf{\sum(f_i)}$ ranges from 0 to 70, i.e.,

$$0 <= \sum(f_i) <= 70$$

and CAF ranges from 0.65 to 1.35 because

a. When $\mathbf{\sum(f_i)} = 0$ then CAF = 0.65
b. When $\mathbf{\sum(f_i)} = 70$ then CAF = 0.65 + (0.01 * 70) = 0.65 + 0.7 = 1.35

Based on the FP measure of software many other metrics can be computed:

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

8. LOCs of an application can be estimated from FPs. That is, they are interconvertible. **This process is known as backfiring**. For example, 1 FP is equal to about 100 lines of COBOL code.

9. FP metrics is used mostly for measuring the size of Management Information System (MIS) software.

10. But the function points obtained above are unadjusted function points (UFPs). These (UFPs) of a subsystem are further adjusted by considering some more General System Characteristics (GSCs). It is a set of 14 GSCs that need to be considered. The procedure for adjusting UFPs is as follows:

a. Degree of Influence (DI) for each of these 14 GSCs is assessed on a scale of 0 to 5. (b) If a particular GSC has no influence, then its weight is taken as 0 and if it has a strong influence then its weight is 5.

b. The score of all 14 GSCs is totaled to determine Total Degree of Influence (TDI).
c. Then Value Adjustment Factor (VAF) is computed from TDI by using the formula: **VAF = (TDI * 0.01) + 0.65**

Remember that the value of VAF lies within 0.65 to 1.35 because

a. When TDI = 0, VAF = 0.65
b. When TDI = 70, VAF = 1.35
c. VAF is then multiplied with the UFP to get the final FP count: **FP = VAF * UFP**

Example: Compute the function point, productivity, documentation, cost per function for the following data:

1. Number of user inputs = 24
2. Number of user outputs = 46
3. Number of inquiries = 8
4. Number of files = 4
5. Number of external interfaces = 2
6. Effort = 36.9 p-m
7. Technical documents = 265 pages
8. User documents = 122 pages
9. Cost = $7744/ month

Various processing complexity factors are: 4, 1, 0, 3, 3, 5, 4, 4, 3, 3, 2, 2, 4, 5.

Solution:

Measurement Parameter	Count		Weighing factor
1. Number of external inputs (EI)	24	*	4 = 96
2. Number of external outputs (EO)	46	*	4 = 184
3. Number of external inquiries (EQ)	8	*	6 = 48

4. Number of internal files (ILF)	4	*	10 = 40
5. Number of external interfaces (EIF) Count-total →	2	*	5 = 378

So sum of all f_i (i ← 1 to 14) = 4 + 1 + 0 + 3 + 5 + 4 + 4 + 3 + 3 + 2 + 2 + 4 + 5 = 43

FP = Count-total * [0.65 + 0.01 *∑(f_i)]
= 378 * [0.65 + 0.01 * 43]
= 378 * [0.65 + 0.43]
= 378 * 1.08 = 408

$$\text{Productivity} = \frac{\text{FP}}{\text{Effort}} = \frac{408}{36.9} = 11.1$$

Total pages of documentation = technical document + user document
= 265 + 122 = 387pages

Documentation = Pages of documentation/FP
= 387/408 = 0.94

$$\text{Cost per function} = \frac{\text{cost}}{\text{productivity}} = \frac{7744}{11.1} = \$700$$

Differentiate between FP and LOC

FP	LOC
1. FP is specification based.	1. LOC is an analogy based.
2. FP is language independent.	2. LOC is language dependent.
3. FP is user-oriented.	3. LOC is design-oriented.
4. It is extendible to LOC.	4. It is convertible to FP (backfiring)