**6.1 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.

Why?

- Helps achieve cost savings by preventing defects
- Helps improve overall project planning
- Facilitates to understand if the desired quality is achieved
- Enforces keenness to further improve the processes
- Helps to analyze the risks associated in a deeper way
- Helps to analyze metrics in every phase of testing to improve defect removal efficiency

**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 , the method of computing metric should be clearly defined.
3. **Applicability**: Metrics should be applicable in the initial phases of development of the software.
4. **Repeatable**: The metric values should be same when measured repeatedly and consistent in nature.
5. **Economical**: Computation of metrics should be economical.
6. **Language Independent**: Metrics should not depend on any programming language.

**Classification:**

Product metrics, Process metrics, and Project metrics.

1. **Product metrics** describe the characteristics of the product such as size, complexity, design features, performance, and quality level.
2. **Process metrics** can be used to improve software development and maintenance. Examples include the effectiveness of defect removal during development, the pattern of testing defect arrival, and the response time of the fix process.
3. **Project metrics** describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

Some metrics belong to multiple categories. For example, the in-process quality metrics of a project are both process metrics and project metrics.

## 6.2 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.

**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

Where, N = Length of program

And estimated program length is denoted by $\widehat{N}$

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

**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

**Purity Ratio** = N^/N
And estimated program length is denoted by N^
N^ = $n1\log_2 n1 + n2\log_2 n2$

**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 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.

$E= D*V$

**Advantages of Halstead Metrics:**

- It is simple to calculate.
- It measures overall quality of the programs.
- It predicts maintenance effort.
- It is useful in scheduling and reporting projects.
- It can be used for any programming language.

**Disadvantages of Halstead Metrics:**

- It depends on the complete code.
- It has no use as a predictive estimating model.

## 6.3 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 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.
5.

**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 ($\eta_2$):** In this metric, the Number of operands used in the program is counted.

  **$\eta_2$ = 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 (LV) is the average live variable metric computed from the ith module. This equation could compute the average span size (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.

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

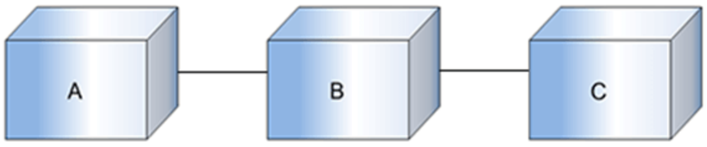
Module Weakness (WM) = $\overline{LV} * \gamma$

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

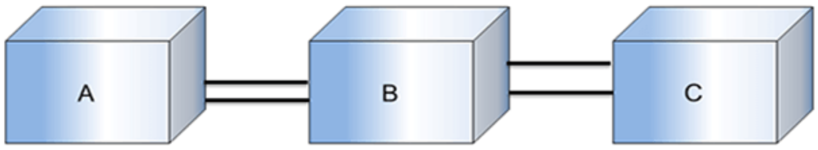
$$\mathbf{WP} = \frac{\left(\sum_{i=1}^{m} \mathrm{WM}_i\right)}{m}$$

Where
**WM$_i$**: Weakness of the ith module
**WP**: Weakness of the program
**m**: No of modules in the program

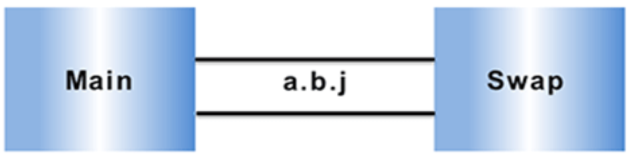
**4.There Sharing of Data among Module:** As the data sharing between the Modules increases (higher Coupling), no parameter passing between Modules also increased, As a result, more effort and time are required to complete the project. So Sharing Data among Module is an important metrics to calculate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



Three modules from an imaginary program



"Pipes"of data shared among the modules



The data shared in program bubble

## 6.4 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.

Procedure Complexity = Length * (FAN-IN * FANOUT)**2

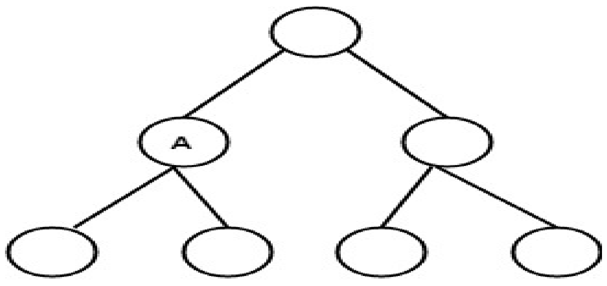


Fig: Aspects of Complexity

## 6.5 Metrics analysis

Metric analysis

You can analyze metrics from multiple perspectives to gain an understanding of past and present performance and to forecast future performance.

You can analyze metrics by using:

- a metrics list
- a metric history
- metric impacts
- diagrams
- reports

**Metrics on a scorecard**
The metrics on a scorecard reveal how well objectives are being met by comparing planned values to actual results. By analyzing the metrics list, you can quickly evaluate performance.

**History charts**
A history chart is a graph that appears for each metric. You can use the history chart to analyze the relationships between target values and actual values.

**Metric impact**
To analyze metric impacts, you must understand your business and the metrics that you are monitoring.

**Diagrams**
A scorecard, metric type, or metric can have one or more diagrams associated with it.

**Reports**
**Reports** found on the Reports tab add information about the strategic value of a scorecard or metric.