Chapter 2

SOFTWARE METRICS

METRIC

Quantitative measure of degree to which a system, component or process possesses a given attribute. "A handle or guess about a given attribute."

E.g. Number of errors found per person hours expended.

Metrics of Project Management can be as follows:

- Budget
- Schedule/Resource Management
- Risk Management
- Project goals met or exceeded
- Customer satisfaction

SOFTWARE METRICS

It refers to a broad range of quantitative measurements for computer software that enable to

- improve the software process continuously
- assist in quality control and productivity
- assess the quality of technical products
- assist in tactical decision-making

Metrics of software products are as follows:

- Focus on Deliverable Quality
- Analysis Products
- Design Product Complexity – algorithmic, architectural, data flow
- Code Products
- Production System

MEASURES

It is the quantitative indication of extent, amount, dimension, capacity, or size of some attribute of a product or process. E.g., Number of errors. Measurement is the act of determining a measure.

Why do we measure?

- To characterize
- To evaluate
- To predict
- To improve

Why do we measure software?

- To predict qualities of a product/process
- To determine the quality of the current product or process.
- To improve quality of the current product or process.
- And to evaluate product/software product

INDICATORS

It is a metric or combination of metrics that provide insight into the software process, a software project, or the product itself. A software engineer collects measure and develops metrics so that indicators will be obtained. An indicator provides insight that enables the project manager or software engineers to adjust the process, the project, or the process to make things better.

SOFTWRAE MEASUREMENT

Software measurement can be divided into two categories.

1. **Direct measures** of SE process include cost and effort. Direct measures of product include LOC produced, execution speed, memory size, and defects reported over some set period of time.
2. **Indirect measures** of product include functionality, quality, complexity, efficiency, reliability, maintainability, and many other "-abilities".

Measurements are done using single or combined metrics.

1. **Size oriented metrics**

It is derived by normalizing quality and/or productivity measures by considering the size of the software that has been produced. For example: choose LOC as normalization value.

Normalization is done by the process by which some measures are developed on the basis of some other metrics.

2. **Function-Oriented Metrics**

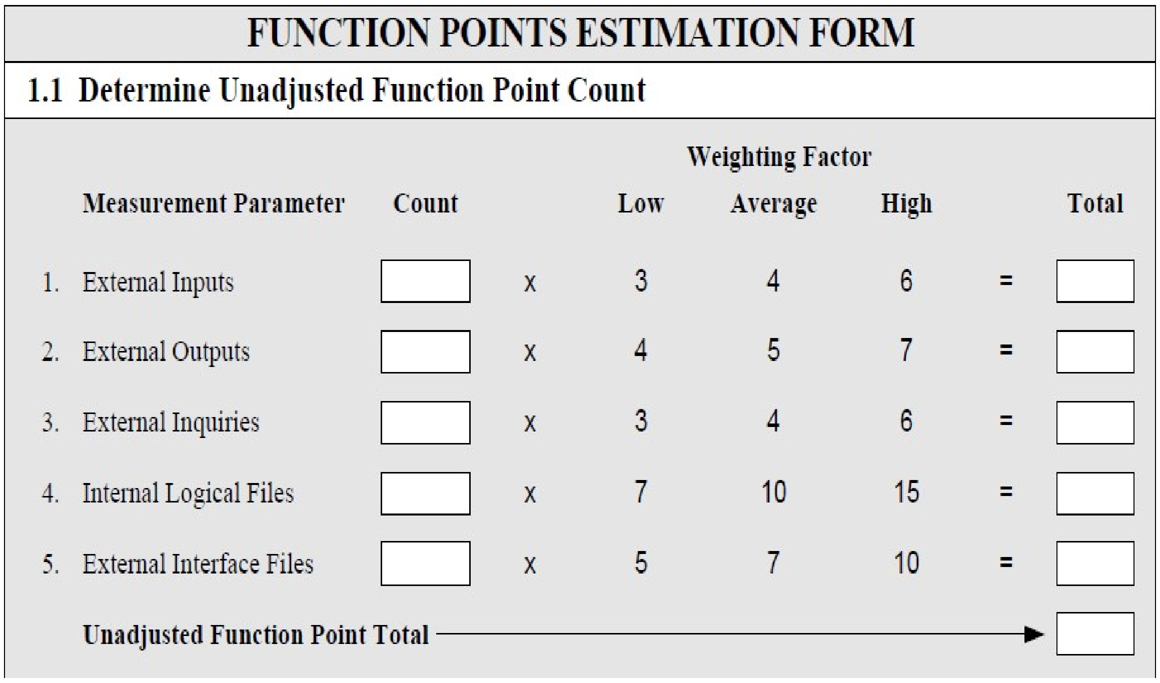
Use a measure of the functionality delivered by the application as a normalization value. The most widely used function-oriented metric is the **Function Point (FP).**

FP measures are used for independent of programming language and it makes it easier to accommodate reuse and trend towards object oriented approaches.

Size of Function Point (FP) = Weighted sum of these five problem characteristics

a. **Number of inputs**: Data items input by user (Group of user inputs taken together).

b. **Number of Outputs**: Reports, Screen outputs, Error Messages
c. **Number of inquiries**: Interactive queries made by users.
d. **Number of Files**: Logical files e.g. data structures, physical files
e. **Number of interfaces**: Interfaces for exchanging information e.g. disk, tapes, communication links.



FUNCTION POINTS ESTIMATION FORM

1.1 Determine Unadjusted Function Point Count

				Weighting Factor			
Measurement Parameter	**Count**		**Low**	**Average**	**High**		**Total**
1. External Inputs		x	3	4	6	=	
2. External Outputs		x	4	5	7	=	
3. External Inquiries		x	3	4	6	=	
4. Internal Logical Files		x	7	10	15	=	
5. External Interface Files		x	5	7	10	=	
Unadjusted Function Point Total →							

- **FP = total unadjusted FP * [0.65+0.01*∑(Fi)]**

  Line of codes and function point are interrelated with respect to programming language.

  Where ∑ (Fi)] is supposed from range <u>42-52</u>

3. **Object oriented metrics**
   Conventional metrics like LOC and FP do not provide enough granularities for the schedule and effort adjustment that are required as we iterate through evolutionary or increment process.

So object oriented projects are suggested some other metrics as:

- Number of scenario script
- Number of key classes
- Number of support classes
- Number of subsystem

4. **Use case oriented metrics**
   It applies use case normalization factor. Use case independent of programming language.

5. **Web engineering project matrices**
   Similar to object oriented conventional metrics doesn’t work here.
   It has some other metrics as:
   - Number of static web pages
   - Number of dynamic web pages
   - Number of internal page link
   - Number of persistent data object
   - Number of executable function.

METRICS FOR SOFTWARE QUALITY

- High quality system development
- Time frame, cost, need satisfaction along with quality.
- Error and defects effects quality
- Metrics such as:

1. Work product errors per functions
2. Errors uncovered per review hour
3. Error uncovered per testing hour

- Error data are used to compute DRE (defect removal efficiency) for each process framework activities.

- Measures are

1. **Correctness :**
   Defects per KLOC (Thousands Line Of Code)
2. **Maintainability:**
   The ease that a program can be corrected, adapted, and enhanced. Time/cost.
   a. Time-oriented metrics: Mean-time-to-change (MTTC)
   b. Cost-oriented metrics: Spoilage – cost to correct defects encountered.

3. **Integrity:**
   Ability to withstand attacks
   a. Threat: the probability that an attack of a specific type will occur within a given time.
   b. Security: the probability that the attack of a specific type will be repelled.

   Integrity = sum [(1 – threat) x (1 – security)]

4. **Usability:**

   Attempt to quantify "user-friendliness" in terms of four characteristics:

   a. The physical/intellectual skill to learn the system

   b. The time required to become moderately efficient in the use of the system

   c. The net increase of productivity

   d. A subjective assessment of user attitude toward the system (e.g., use of questionnaire).

DEFECT REMOVAL EFFICENCY:

DRE is defined as:

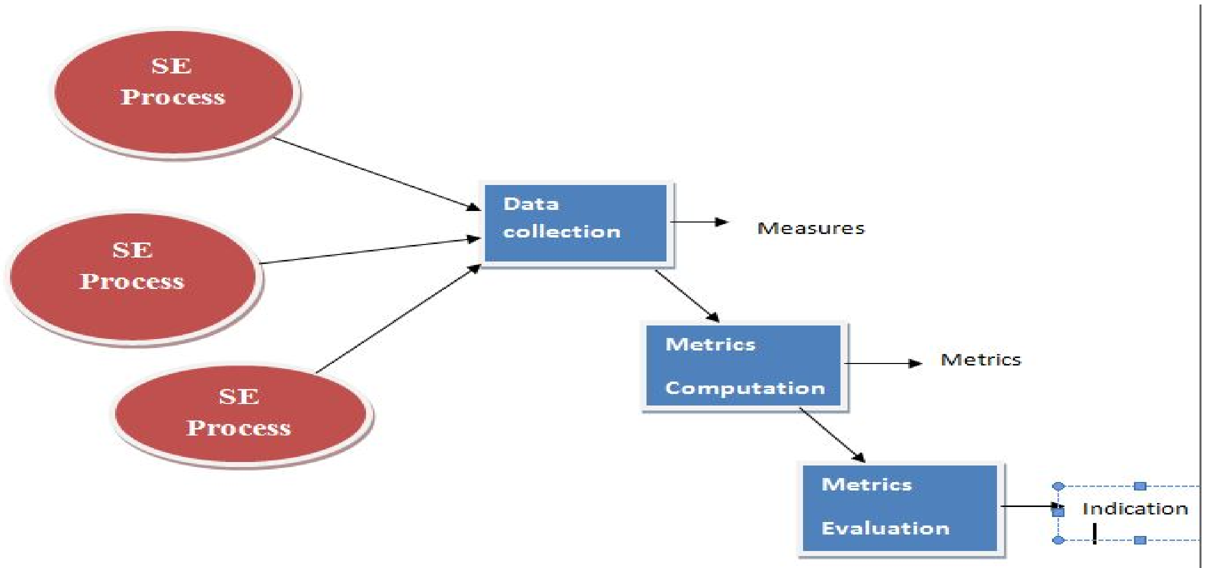
$$DRE = E/ (E + D)$$

Where E is the number of errors found before delivery of software to the end-user and D is the number of defects found after delivery

The ideal value for DRE is 1 → no defects found.

STATICAL QUALITY CONTROL:

- “Why do we need so?” is the general question.
- If we measure, we will know we are improving and if we are not improving we are lost.
- It helps in setting meaningful goals.
- Measurement is used to establish a process baseline from which improvements can be assessed.
- Helps in managing project estimates producing highest quality and do everything on time.
- Helps in removal of “vital few” causes of defects that have the greatest impact on the software development.
- Statistical quality control focuses on use of metrics of previous projects to control quality of the upcoming one.



METRICS FOR SMALL ORGANIZATION:

- The vast majority of software development organization (small organization) has fewer than 20 software people. So it is unreasonable and in most cases unrealistic to expect that such organizations will develop comprehensive software metrics programs.
- Keep it simple is a guideline that works reasonably well in many activities so simple software metrics is derived which still provides value.
- A small organization might select the following set of easily collected measures.

   a. Time (hours or days)elapsed from the time a request is made until evaluation is complete, tqueue
   b. Effort (person, hours) to perform the evaluation (Weval), to make change (Wchange)
   c. Time required (days/hours) to make change, tchange
   d. Errors uncovered during work to make change, Echange
   e. Defects uncovered after product is released to the customer, Dchange

   Defect removal efficiency (DRE) = Echange/ (Echange + Dchange)