



Chapter 2: Software Metrics





Examples of Metrics from Everyday Life

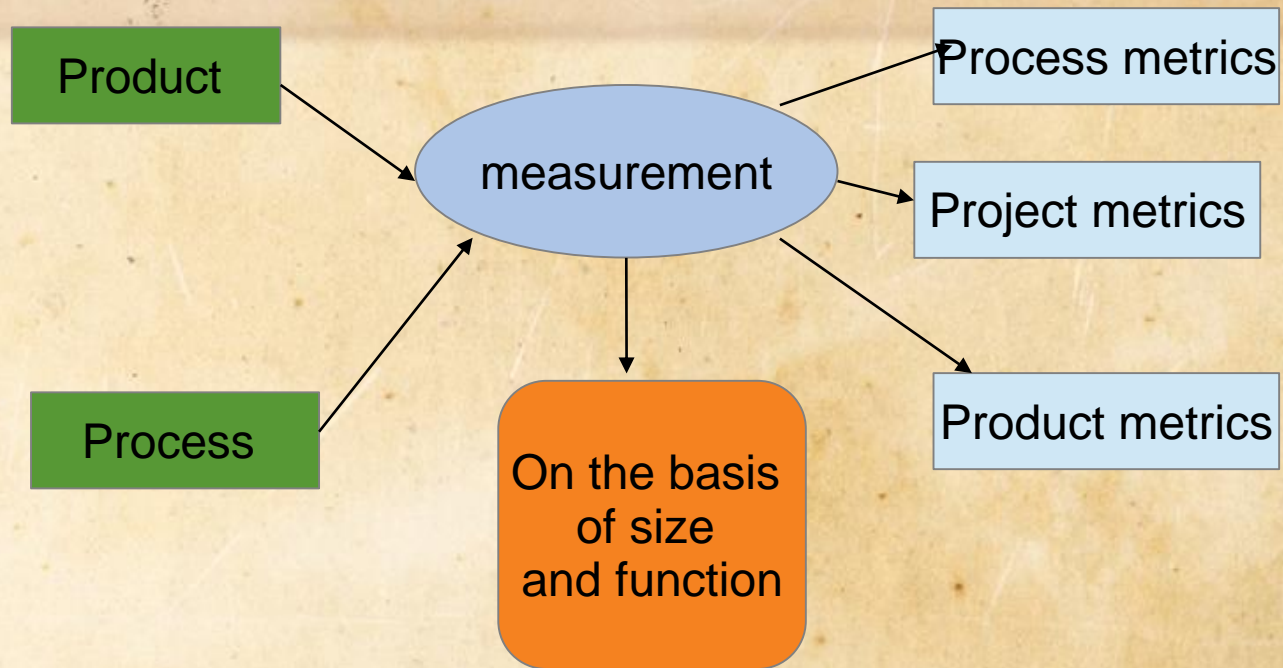
- **College experience**
- Grades received in class last semester
- Number of classes taken each semester
- Amount of time spent in class this week
- Amount of time spent on studying and homework this week
- Number of hours of sleep last night

Why we require metrics

- Software measures can be understood as a process of quantifying and symbolizing various attributes and aspects of software
- Software provides measures for various aspects of software process and software product.
- Software measures are fundamental requirements of software engineering. They not only help to control the software development process but also aid to keep the quality of ultimate product excellent.
- According to the Tom DeMarco (Software Engineer), **you can not control what you cannot measure**. By his saying it is very clear how important the software measures are.

- 
- A key element of any engineering process is measurement.
 - We can use measures to better understand the attributes of the models that we create and to assess the quality of the engineered products or systems that we build.
 - But unlike other engineering disciplines, software engineering is not grounded in basic quantitative law of physics.
 - Software measures and metrics are indirect.
- 

- A good manager should:



Product metrics

Describes the characteristics of the product.

- **Size**
- **Complexity**
- **Design features**
- **Performance**
- **Quality level**
- **Reliability**
- **functionality**

Process metrics

These characteristics can be used to improve the development and to improve
The development and maintenance activities of the software

- Efforts required in the product
 - Time to produce the product
- Effort of development of technical tools
 - No. of defects found during testing
 - Productivity
 - Quality
 - Failure rate

Project metrics

It describes the project characteristics and execution

- no. of software developers
- Staffing pattern over the life cycle of the software
 - Cost
 - Schedule
 - productivity

Measures, Metrics, and Indicators



- These three terms are often used interchangeably, but they can have subtle differences

Measure

Provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process

Measurement


The act of determining a measure

Metric

(IEEE) A quantitative measure of the degree to which a system, component, or process possesses a given attribute

Indicator

A metric or combination of metrics that provides insight into the software process, a software project, or the product itself



Software measurement



- **Direct measures:**


easy to collect

E.g. cost, effort, line of codes (Loc), execution speed, memory size, defects etc

- **Indirect measures:**

more difficult to assess and can be measured indirectly only.

Quality, functionality, complexity, reliability, efficiency, maintainability etc



An Example:

2 different project teams are working to record errors in a software process

Which team do you think is more effective in finding errors?

Team A: finds 342 errors
During software process
Before release



Team B:
Finds 184 errors



Normalization of Metrics:

To answer this we need to know the size & complexity of the projects.

But if we normalize the measures, it is possible to compare the two

For normalization we have 2 ways-

Size-Oriented Metrics

Function Oriented Metrics

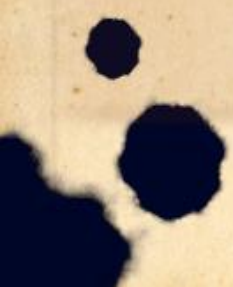
Size oriented metrics

Based on the “size” of the software produced

project	Efforts (person-month)	cost	LOC	KLOC	Doc pgs	errors	people
A	24	1,68,000	12100	12.1	365	29	3
B	62	440,000	27200	27.2	1224	86	5



From the above data simple size oriented metrics can be developed for each project

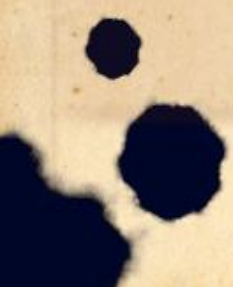
- **Errors per KLOC**
 - **Cost per KLOC**
 - **Pages of document per KLOC**
 - **Errors per person-month**
 - **LOC per person-month**
- 



Advantages

- LOC can easily counted
- Many software estimation models uses LOC or KLOC as input

Disadvantages

- LOC measures are language dependent, programmer dependent.
 - Their use in estimations requires a lot of details which can be difficult to achieve.
- 

Function-oriented metrics

- Based on “functionality” delivered by the software
- Functionality is measured indirectly using a measure called function point .
- Function points (FP) - derived using an empirical relationship based on countable measures of software’s information domain & assessments of software complexity

Steps in calculating FP

1. Count the measurement parameters.
2. Assess the complexity of the values.
3. Calculate the raw FP (see next table).
4. Rate the complexity factors to produce the complexity adjustment value(CAV).
5. Calculate the adjusted FP as follows:

$$\text{FP} = \text{raw FP} \times [0.65 + 0.01 \times \text{CAV}]$$

Information domain values are defined in the following manner:

Measurement Parameter	Examples
1. Number of external inputs (EI)	Input screen and tables.
2. Number of external outputs (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.

Measurement parameter	Count	Weighting factor				
		Simple	Average	Complex		
Number of user inputs	<input type="text"/>	x	3	4	6	= <input type="text"/>
Number of user outputs	<input type="text"/>	x	4	5	7	= <input type="text"/>
Number of user inquiries	<input type="text"/>	x	3	4	6	= <input type="text"/>
Number of files	<input type="text"/>	x	7	10	15	= <input type="text"/>
Number of external interfaces	<input type="text"/>	x	5	7	10	= <input type="text"/>
Count total						<input type="text"/>

Rate complexity factor

For each complexity adjustment factor, give a rating on scale 0 to 5


0	No influence
1	incidental
2	moderate
3	average
4	significant
5	essential

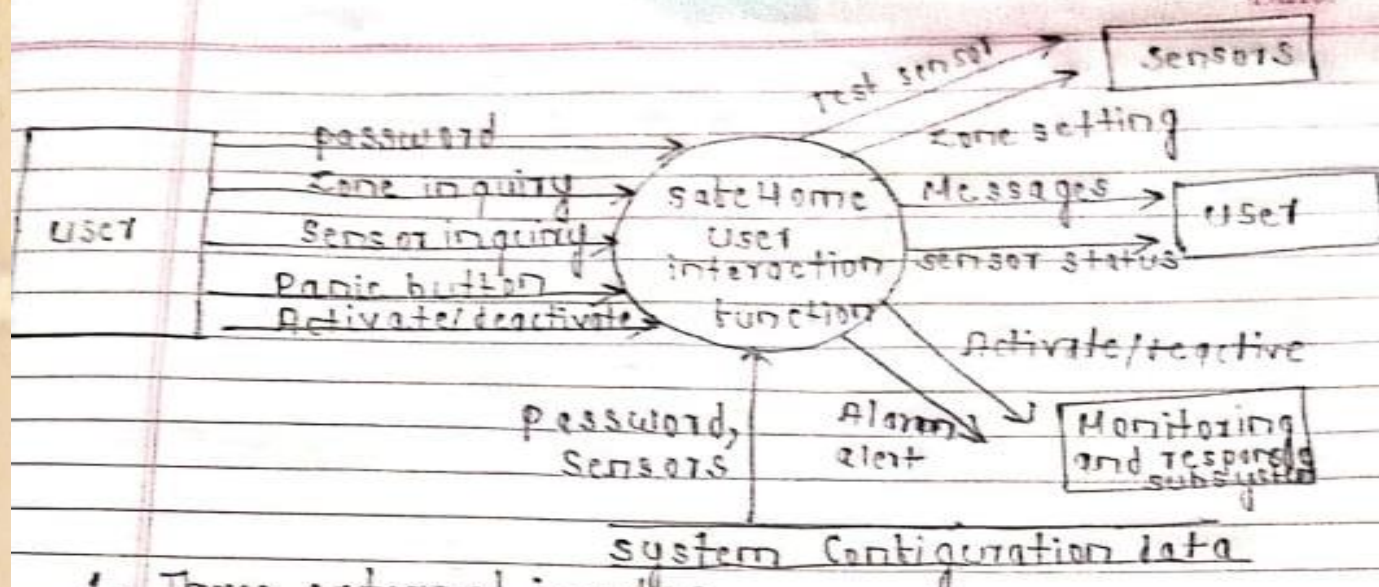
Complexity adjustment factors

The software complexity can be computed by answering the following questions :

The rating for all the factor F1 TO F14 are summed to produce the complexity adjustment value(CAV)

- 1.Does the system need reliable backup and recovery?
- 2.Are data communications required?
- 3.Are there distribute processing functions?
- 4.Is the performance of the system critical?
- 5.Can the system be able to run in an existing, heavily, and largely utilized operational environment?
- 6.Does the system require on-line data entry?

- 
- 7.Does the input transaction is required by the on-line data entry to be built over multiple screens or operations?**
 - 8.Are the master files updated on-line?**
 - 9.Are the inputs, outputs, files, or inquiries complex?**
 - 10.Is the internal processing complex?**
 - 11.Is the code which is designed to be reusable?**
 - 12.Are conversion and installation included in the design?**
 - 13.Is the system designed for multiple installations in various organizations whenever required?**
 - 14.Is the application designed to facilitate or make the change and provide effective ease of use by the user?**




1. Three external inputs:-
password, panic button, activate/deactivate
2. Two external inquiry → zone inquiry and sensor inquiry
3. one internal logic tiles → system configuration tile
4. Two external output → Messages and sensor status
5. Four External interface files → test sensor, zone setting, activate/deactivate and alarm alert.

Date: / /

Information domain value	count	weighting factor		
		simple	Average	complex
External inputs (EIS)	3	(3)	4	6 = 9
External outputs (EOS)	2	(4)	5	7 = 8
External Inquiries (EQs)	2	(3)	4	6 = 6
Internal logical files (ILFs)	1	(7)	10	15 = 7
External interface files (EIFs)	4	(5)	7	10 = 20
count total		→ 50		

Now,

$$FP = 50 \times [0.65 + (0.01 \times 46)] = 56$$



After calculating the function point, various other measures can be calculated as shown below :

Productivity = $\text{FP} / \text{person-month}$

Quality = $\text{Number of faults} / \text{FP}$

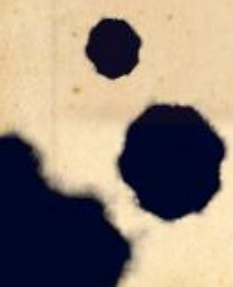
Cost = $\text{\$/FP}$

Documentation = $\text{Pages of documentation} / \text{FP}$





Example of Function-Oriented Metrics

- Errors per FP
 - Defects per FP
 - \$ per FP
 - Pages of documentation per FP
 - FP per person month
- 



Advantages:

language independent, based on data known early in project, good for estimation

Disadvantages:

calculation complexity, subjective assessments, FP has no physical meaning (just a number)

