Chapter 2: Software Metrics

Software Metrics

- Software metrics are quantifiable measures of software characteristics, used to assess quality, progress, and health throughout the development lifecycle.
- They are crucial for cost estimation, productivity measurement, performance optimization, testing, and project management, and are generally categorized as product, process, or project metrics.

Examples of Metrics from Everyday Life

College experience

- **Grades received last semester** – a metric that tells how well you did in your courses.
- **Number of classes taken each semester** – shows how heavy your course load is.
- **Time spent in class this week** – measures how many hours you actually sat in lectures/labs.
- **Time spent on studying and homework this week** – shows how much effort you put in outside class.
- **Hours of sleep last night** – a metric about your health and rest.

Why we require metrics?

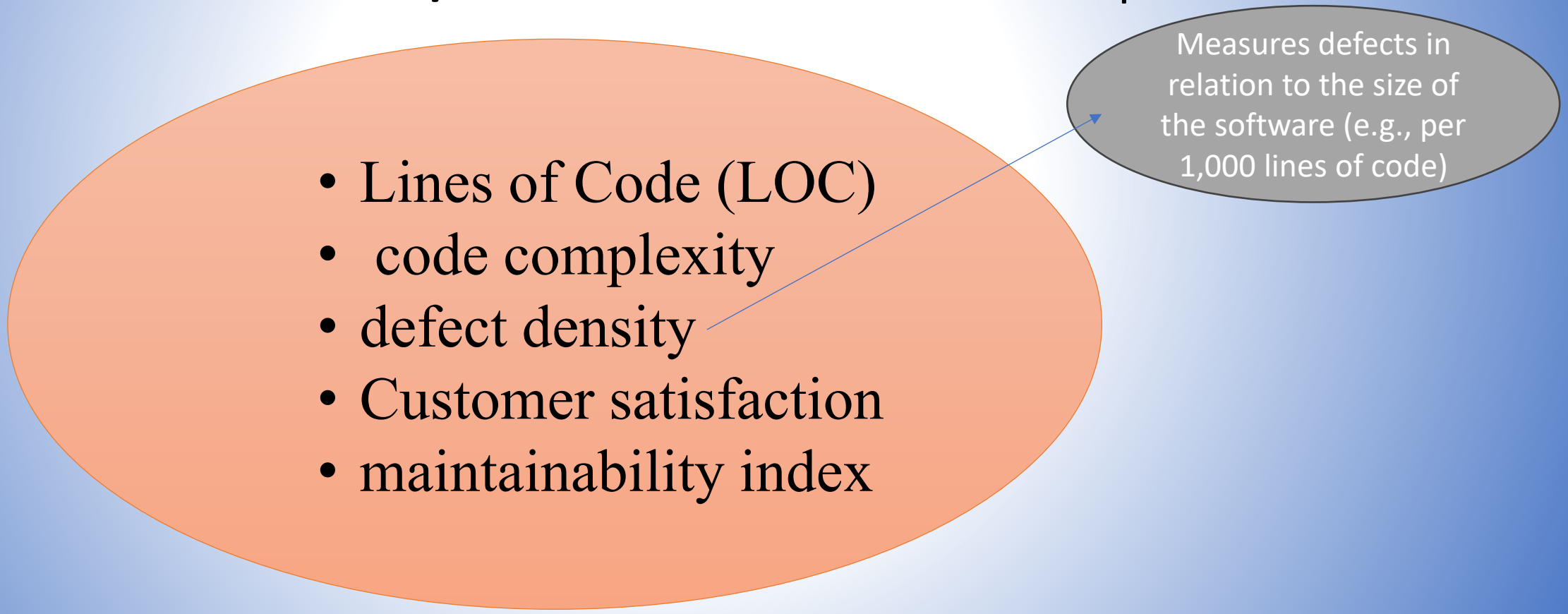
- Software measures can be understood as a process of quantifying and symbolizing various attributes and aspects of software. Software provides measures for various aspects of software process and software product.
- Software measures are fundamentals requirements of software engineering. They not only 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.(because they assess software qualities like complexity, reliability, and maintainability by using related parameters, rather than a direct scale)

Types of Software Metrics

Product metrics

- Measure the **attributes/characteristics** of the software product itself.



Process Metrics

- These measure the **efficiency and effectiveness of the development** methodologies, techniques, and tools used.

It highlights numerous factors such as fault detection, techniques, tools, defects or error efficiency, and overall software reliability.

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

Measures

- A single, quantitative indication of an attribute of a software product or process.
- It is a number or a quantity that records a directly observable value.
- Example: The direct count of errors found in a specific hour (\(10\) errors) or the number of lines of code in a file.

Measures, Metrics, and Indicators

Measures

- Example: 5 defects found in module A.
- Example: 2,000 lines of code in a file.

These are simple counts or amounts, with no extra calculation.

Measures, Metrics, and Indicators

Metrics

- A quantitative measure of the degree to which a system, component, or process possesses a given attribute.
- Often derived from one or more measures and used to assess and compare attributes.
- **Examples:**
  - **Defect density:** Defects per thousand lines of code.
  - **Cyclomatic complexity:** A measure of code complexity.
  - **Mean Time to Recovery (MTTR):** Average time it takes to restore the system after a failure.

Measures, Metrics, and Indicators

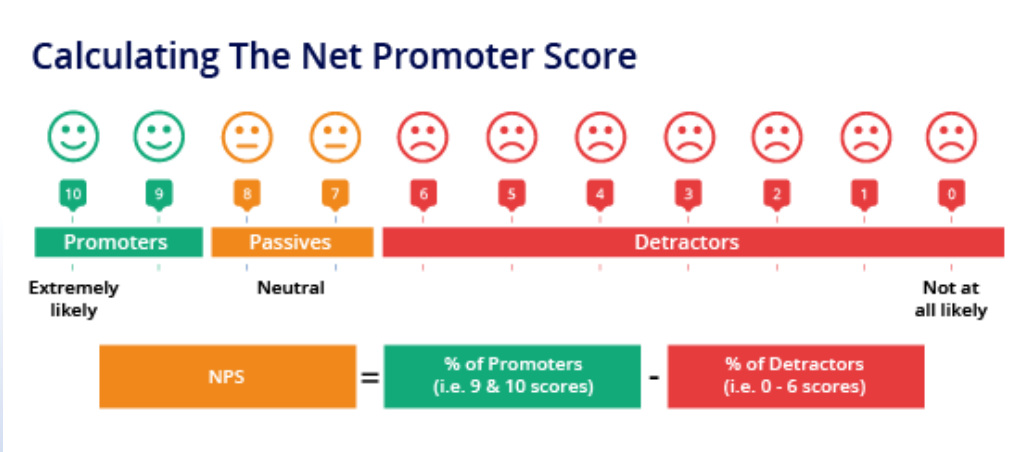
Metrics

- Example: Defect density = number of defects / KLOC (thousand lines of code), e.g., 5 defects / 2 KLOC = 2.5 defects per KLOC.
- Example: Test coverage = (number of executed test statements / total statements) × 100, e.g., 800 / 1,000 = 80%.

Metrics help compare things (between modules, releases, teams, time periods).

Indicators

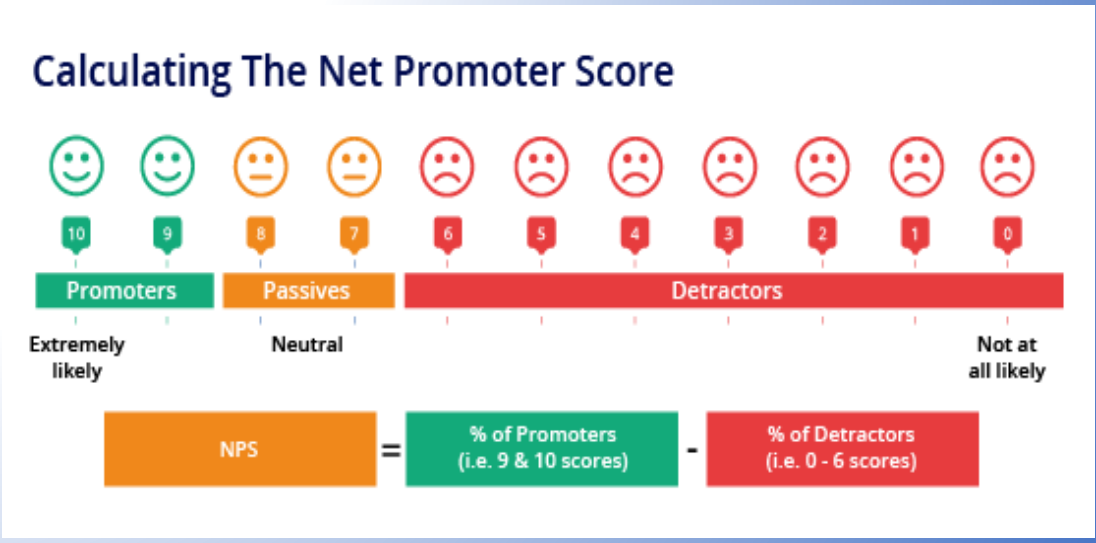
- A metric or combination of metrics that provides insight into the software process, a software project, or the product itself
- Used to provide meaningful and timely information to guide management and engineering decisions.
- **Example : Customer satisfaction indicator**



Indicators

- Example: If defect density > 5 defects/KLOC, the module is marked “high-risk” and needs extra testing.
- Example: If test coverage drops below 70%, the indicator says “insufficient testing” and the team must add more tests.

So, indicators turn metrics into clear messages for managers and teams.



Software measurement

Direct Measurement:

- This involves measuring a product, process, or attribute directly with a standard scale.

  - **Examples:** Calculating the physical size of the code or measuring the time it takes to complete a task.

Software measurement

Indirect Measurement:

- This method measures a quality by using related parameters or references.
  - **Examples:** Measuring code quality, reliability, or performance by analyzing attributes like maintainability and complexity.

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.

Size-oriented metrics are numbers that describe a project mainly by how *big* the software is (in LOC, documents, etc.) and then relate that size to effort, cost, errors, and people.

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

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 a table you can calculate simple size-oriented metrics like:

- Effort per KLOC = effort / KLOC → shows how many person-months are needed for each KLOC.
- Cost per KLOC = cost / KLOC → shows how expensive each KLOC is.
- Errors per KLOC = errors / KLOC → shows code quality (fewer errors per KLOC is better).
- LOC per person = LOC / people → shows average productivity per developer.

Size oriented metrics

Hence, These metrics let you compare projects A and B even if they are different sizes, because you normalize everything by size (KLOC).

That helps in planning new projects, estimating required effort and cost, and checking productivity and quality.

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.

Lines of code

1. Cost per line of code = Labor rate / Productivity
2. Estimated project cost = Estimated line of code*Cost per line of code
3. Estimate labor effort = Estimated line of code / Productivity

Example Of LOC

- Estimated line of code = 33,200
- Productivity = 620 LOC/PM
- Labor Rate = $ 8000/PM
- Cost per line of code = ?
- Estimated Project cost = ?
- Estimate Labor Effort = ?

Example Of LOC

- Cost per line of code = $8000/620
  = $13
- Estimated project cost = 33,200 * 13
  = $431,600
- Estimated Labor effort = LOC / productivity
  = 33,200/620
  = 54 PM

Function-oriented metrics

- Function-oriented metrics measure software size based on its functionality from the user's perspective, with Function Points (FP) being the most popular metric.
- Functionality is measured indirectly using a measure called function point .
- Function points (FP) - derived using an empirical relationship(observation) 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: FP = raw FP x [0.65 + 0.01 x 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.

List of 5 Unadjusted Function Points

1) External Inputs (EI): 3
•User registration form, product search form, add to cart form.

2) External Outputs (EO):
•Confirmation email, order summary page, invoice generation.

3) External Inquiries (EQ):
•Product availability check, order status check.

4) Internal Logical Files (ILF):
•User data, product catalog, order data.

5) External Interface Files (EIF):
•Payment gateway interface, third-party shipping service interface.

Measurement parameter	Count		Weighting factor				
			Simple	Average	Complex		
Number of user inputs		x	3	4	6	=	
Number of user outputs		x	4	5	7	=	
Number of user inquiries		x	3	4	6	=	
Number of files		x	7	10	15	=	
Number of external interfaces		x	5	7	10	=	
Count total							

Function Point (FP) Metrics

Weighting Factor

Parameter	Count		Simple	Average	Complex		
Inputs		x	3	4	6	=	
Outputs		x	4	5	7	=	
Inquiries		x	3	4	6	=	
Files		x	7	10	15	=	
Interfaces		x	5	7	10	=	
					Count-total (raw FP)		

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

Factor	Value
Backup and recovery	4
Data communications	2
Distributed processing	0
Performance critical	4
Existing operating environment	3
Online data entry	4
Input transaction over multiple screens	5
Master files updated online	3
Information domain values complex	5
Internal processing complex	5
Code designed for reuse	4
Conversion/installation in design	3
Multiple installations	5
Application designed for change	5
Value adjustment factor	**1.17**

Complexity adjustment factors

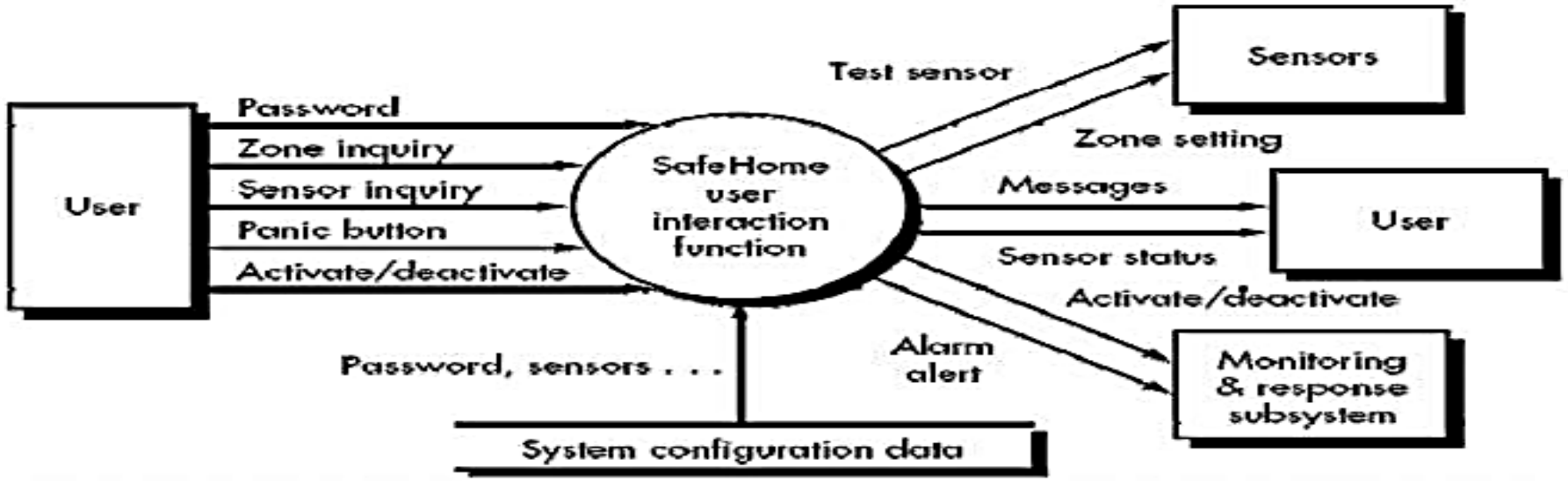
The software complexity can be computed by answering the following questions :

(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?

Complexity adjustment factors

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?



Weighting Factor

Measurement parameter	Count		Simple	Average	Complex		
Number of user inputs	3	×	3	4	6	=	9
Number of user outputs	2	×	4	5	7	=	8
Number of user inquiries	2	×	3	4	6	=	6
Number of files	1	×	7	10	15	=	7
Number of external interfaces	4	×	5	7	10	=	20
Count total							50

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

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

(0.01 * 14*3(average)

Where
FP=Functional point
CAV=Complexity adjustment value(Total Degree of Influence of the 14 General System Characteristics)

$$FP = raw\ FP \times [0.65 + 0.01 \times CAV]$$
$$= 50*[0.65+0.01*(14*3)]$$
$$= 53.5$$

- After calculating the function point, various other measures can be calculated as shown below :
- Productivity = FP / person-month
- Quality = Number of faults / FP
- Cost = $ / FP
- Documentation = Pages of documentation / FP

Formula for Function Point

- Cost per FP = labor rate/productivity
- Estimated project cost = estimated fP* cost per Fp
- Estimated Labor effort= Estimated FP/Productivity

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

Consider a software project with the following information domain characteristic for the calculation of function point metric.

FP = UFP * VAF

Number of external inputs (I) = 30

Number of external output (O) = 60

Number of external inquiries (E) = 23

Number of files (F) = 08

Number of external interfaces (N) = 02

It is given that the complexity weighting factors for I, O, E, F, and N are 4, 5, 4, 10, and 7, respectively. It is also given that, out of fourteen value adjustment factors that influence the development effort, four factors are not applicable, each of the other four factors has value 3, and each of the remaining factors has value 4. The computed value of the function point metric is _____.

Parameter	count		weighting factor complex	
Input (I)	30	X	4	= 120
Output (O)	60	X	5	= 300
inquries (E)	23	X	4	= 92
Files (F)	08	X	10	= 80
Interfaces (N)	02	X	7	= 14

count Total (raw FP) = 606

Influence of 14 factors = (4 X 0) + (4 X 3) + (6 X 4)
= 0 + 12 + 24
= 36

∴ Functional point = raw Fp X CAF
(UFP X VAF)

= 606 X [0.65 + 0.01 X 36]
= 612 #

productivity = $\frac{612}{36}$ = 17
(effort = 36P-M)

Example :

With the given data for an online shopping site developed by ABC software developers

- Number of user input = 98
- Numbers of user Output = 51
- Number of User Inquires = 47
- Number of External Interfaces = 32
- Number of Logical Files = 61

- Assuming that the complexity of the given website development is average, compute the function point if the productivity of the ABC S/W developers is 35 FP/PM and their salary structure is Rs. 1500 per month on average, estimate total cost of the software

Step 1: Calculate Unadjusted Function Points (UFP)

- Assign the correct weights:
- User Input (EI): 98 × 4 = 392
- User Output (EO): 51 × 5 = 255
- User Inquiry (EQ): 47 × 4 = 188
- Logical Files (ILF): 61 × 10 = 610
- External Interface (EIF): 32 × 7 = 224
- Sum:
- $\text{UFP} = 392 + 255 + 188 + 610 + 224 = 1669$

Step 2: Complexity Adjustment Factor (CAF)

- Average complexity (all factors at 3):
- $F = 14 \times 3 = 42$
- $\text{CAF} = 0.65 + 0.01 \times 42 = 1.07$

Step 3: Adjusted Function Points (FP)

- $\text{FP} = 1669 \times 1.07 = 1785.83$

Step 4: Effort in person-months

- Productivity = 35 FP/PM:
- Effort $= \frac{1785.83}{35} \approx 51.02$ *person−months*

Step 5: Total cost $=$ *Effort* $\times$ *salary per month*

- Cost $= 51.02 \times 1{,}500 = 76{,}530$ *Rs*

Final Answers:

- Function Points (FP): 1785.83
- Total Effort: 51.02 person-months
- Estimated Cost: Rs. 76,530

Practice question:

Assuming that the complexity of the NCIT MIS software development is average, compute the function point for it with the given data:

Number of User Input : 95

Number of User Output : 55

Number of Inquiries : 4

Number of logical Files : 66

Number of External Interfaces:27

If the productivity of the software developers is 30 FP/PM and their salary structure is RS.18000 per month on average, estimate the total cost of the software.

Solution:

First compute function points, then effort and cost.

1. Unadjusted Function Points (average complexity)

For *average* complexity, typical weights are:

- External Inputs (EI): 4 FP each
- External Outputs (EO): 5 FP each
- External Inquiries (EQ): 4 FP each
- Internal Logical Files (ILF): 10 FP each
- External Interface Files (EIF): 7 FP each

- Given data:
- User Inputs (EI) = 95 → $95 \times 4 = 380$ FP
- User Outputs (EO) = 55 → $55 \times 5 = 275$ FP
- Inquiries (EQ) = 4 → $4 \times 4 = 16$ FP
- Logical Files (ILF) = 66 → $66 \times 10 = 660$ FP
- External Interfaces (EIF) = 27 → $27 \times 7 = 189$ FP
- Total (Unadjusted FP, assuming average complexity and no VAF given):
- $FP = 380 + 275 + 16 + 660 + 189 = 1520$

2. Effort in person-months

- Productivity = 30 FP per person-month (FP/PM).
- Effort $= \frac{FP}{Productivity} = \frac{1520}{30} \approx 50.67\ PM$
- So about 51 person-months of effort.

- **3. Total cost**
- Average salary = Rs. 18,000 per month.
- Cost $= Effort \times salary\ per\ month = 50.67 \times 18{,}000 \approx 912{,}000\ Rs$
- So the estimated development cost is about Rs. 9.1 lakhs (≈ Rs. 9,12,000).

(2024 spring question no.1 b)

Given the data below, compute the function point value, productivity, documentation and cost per function for a project with the following information domain characteristics.

- Number of user inputs: 28
- Number of user outputs: 44
- Number of user inquiries: 7
- Number of files: 3
- Number of external interfaces: 2

and Effort=37P-M, Technical document=360 pages, user document=129 pages, cost=Rs 8000 per month complexity adjustment values are 4,1,1,3,5,5,4,4,3,3,2,3,4,5

Q. No 1(b)

Given. Number of user inputs = 28
Number of user outputs = 44
Number of user inquiries = 7
Number of files = 3
Number of external interfaces = 2
Effort = 37 P-M
Technical document = 360 pages
user document = 129 pages
cost = Rs 8000 P-M

Information domain	count	weighting factor	FP count
no. of user inputs	28	4	112
no of user outputs	44	5	220
no of user inquiries	7	4	28
no of files	3	10	30
no of external interface	2	7	14
			404

Raw FP (count total) = 404

$\Sigma Fi = 4+1+1+3+5+5+4+4+3+3+2+3+4+5$
$= 47$

function point (FP) : Raw FP $\times [0.65 + (0.01 \times \Sigma Fi)]$
$= 404 \times [0.65 + (0.01 \times 47)]$
$= 452.48$
≈ 453

Productivity = $\frac{FP}{effort} = \frac{453}{37} = 12.24$ / P-M

Total pages of documentation = 360 + 129
= 489

Documentation = $\frac{\text{Pages of documentation}}{FP}$
$= \frac{489}{453}$
$= 1.0794$

Cost per function = $\frac{\text{cost}}{\text{productivity}}$
$= \frac{8000}{12.24}$
= Rs 653.59 per function

Metrics For Software Quality

What is software quality?

Quality software is reasonably bug or defect free, delivered on time and within budget, meets requirements or expectation , and Is maintainable.

Software quality can be measured through the software Engineering process, before release to customer and after release to the customer.

A good software Engineer and good software engineering must measure if high quality is to be realized .

Key aspects of quality for the customer includes:

Good design :looks and style Good
Functionality : it does the job well
Reliable: acceptable level of breakdowns or failure consistency
Durable : last as long as it should Good after sales service
Value for money

Measuring quality

- Although there are many measures of software quality **Correctness**, **maintainability**, **integrity** and **usability** provide useful indicators for the project team

Correctness

- Is the degree to which the software performs its required function
- Defects are those problems reported by user after the programs has been released for general use.
- For quality assessment purposes, defects are counted over a standard period of time, typically one year.
- The most common measure for correctness is defects per KLOC

(Defects lack of conformance to requirements)

Maintainability

- Is the ease with which program can be corrected if an error is encountered, adapted if its environment changes, or enhanced if the customer desires a change in requirements.
- No way to measure directly so we must use indirect measure.
- A simple time-oriented metric is mean-time-to-change(MTTC)

the time it take to analyze change request,design an appropriate modification, implement the change, test it and distribute the change to all users.

Integrity:

- Measures a systems ability to withstand attacks(both accidental and intentional) to its security.
- To measure integrity we should know threat and security.
- Threat is the probability(estimated or derived from empirical evidence)that can attack of a specific type will occur within a given time.
- Security is the probability that the attack a specific type will be repelled.
- Integrity =Σ[1-(threat*(1-security))]

Example:threat=0.25 and security=0.95
Then integrity=0.99(very high)
If,threat=0.50
Security=0.25
Integrity=0.63(unacceptably low)

Usability

It indicates the usability i.e., the extent to which the user's feel the software as easy to use.

A quality metric that provides benefit at both the project and process level is **defect removal efficiency (DRE)**

DRE is computed as

DRE = E/(E+D)

Where E = Errors found before delivery of the software

D = Defects found after the delivery

- Ideal value for DRE is 1.(i.e no defects are found in the software)
- As E increases, it is likely that the final value of D will decrease(errors are filtered out before they become defects.
- DRE encourages a software project team to find as many errors as possible before delivery.
- DRE can also be used within the project to assess a teams ability to find errors before they are passed to the next frame work activitiy.
- DREi=Ei/(Ei+Ei+1),where Ei is the number of errors found during software engineering action i and Ei+1 is the number of errors found during software engineering action i+1
- A quality objective for a software team(or an individual software engineer) is to achieve DREi that approaches 1.

Metrics for small organizations

- The majority of software development organizations have **fewer than 20 software people.**
- It is unreasonable, and in most cases **unrealistic to expect** that such organizations **will develop comprehensive software metrics programs.**
- However, they must focus on metrics to help improve their local software process, quality and timeliness of the product they produce.
- A small organization can **begin by focusing not on measurement but rather on results.**
- The software group can define a single objective that requires improvements.
- **For example Reduce the time to evaluate and implement change requests. A small organization might select the following set of easily collected measures.**

- Tqueue: time(hours or day) elapsed from the time request is made until evaluation is completed
- Weval: effort(person-hours)to perform the evaluation
- Teval: time(hours or days)elapsed from completion of evaluation to assignment of change order to personnel.
- Wchange: effort(person-hours) required to make the change.
- Tchange: time required(hours or days) to make the change.
- Echange: errors uncovered during work to make change(internal defects) .
- Dchange: defects uncovered after changes is released to the customer (external defects).

Once these measures have been collected for a number of change requests , we can calculate

The total time from when a change is requested until it is finished (implemented).

The percentage of that total time spent specifically on:

- Waiting in the initial queue.
- Being evaluated and assigned to a team.
- The actual work of implementing the change