

Process and Project Metrics

Software metrics

Software metrics (process and project) are quantitative measures

Measurement can be applied

- To the software process with the intent of improvement
- To assist in estimation, quality control, productivity assessment, and project control
- To help assess the quality of technical work products
- To assist in tactical decision making as a project proceeds

Metric and Indicator

- A **metric** is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.
- An **indicator** 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 measures and develops metrics so that indicators will be obtained .

Process Metrics

- Process metrics are collected across all projects and over long periods of time.
- Process metrics provide a set of process indicators that lead to long-term software process improvement

Project metrics

Enable a software project manager to

- Assess the status of an ongoing project
- Track potential risks
- Uncover problem areas before they "go critical"
- Adjust work flow or tasks

Project metrics

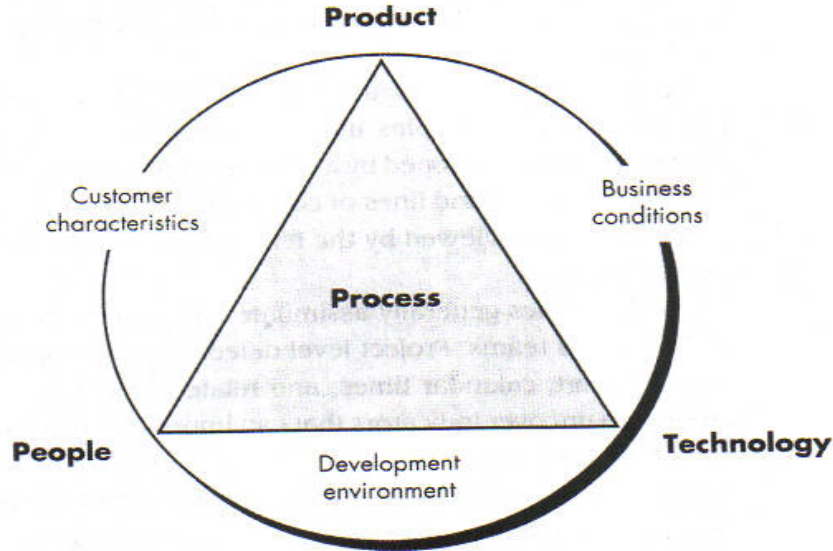
- Evaluate the project team's ability to control quality of software engineering work products.
- Estimate effort and time duration
- Every project should measure input, output and result

Process Metrics and Software Process Improvement

The only rational way to improve any process is

- To measure specific attributes of the process
- Develop a set of meaningful metrics based on these attributes
- Use the metrics to provide indicators that will lead to a strategy for improvement

Determinants for Software quality & Organizational Effectiveness



Private and Public metric

There are "private and public" uses for different types of process data

- Data *private* to the individual
 - Serve as an indicator for the individual only
- Eg : Defect rates, Errors found during development

Public metric

- Defects reported for major software functions
- Errors found during formal technical reviews
- Lines of code or function points per module/function

Software Measurement

- Direct measures
 - ❑ Software process (cost)
 - ❑ Software product (lines of code (LOC), execution speed, defects reported over some set period of time)
- Indirect measures
 - ❑ Software product (Functionality, quality, complexity, efficiency, reliability, maintainability)
- Many factors affect software work, it is difficult (don't use metrics) to compare individuals/team

Size-Oriented Metrics

- Derived by normalizing quality and/or productivity measures by considering the "size" of the software
- Metrics include
 - Errors per KLOC, Defects per KLOC, Dollars per KLOC, Pages of documentation per KLOC

Project	LOC	Effort	\$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
•	•	•	•	•	•		
•	•	•	•	•	•		
•	•	•	•	•	•		

Function-Oriented Metrics

- It is a measure of the functionality delivered by the application as a normalization value
- Eg. Number of input, number of output, number of files, number of interfaces

Reconciling LOC & FC Metrics

- The relationship between lines of code and function points depends programming language
- Rough estimates of the average number of lines of code required to build one function point in various programming languages

Programming Language	LOC per Function point			
	Avg.	Median	Low	High
Access	35	38	15	47
Ada	154	—	104	205
APL	86	83	20	184
ASP 69	62	—	32	127
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
Clipper	38	39	27	70
COBOL	77	77	14	400

Metrics in the Process Domain

Metrics in the Process Domain

- Process metrics are collected across all projects and over long periods of time
- They are used for making strategic decisions
- The intent is to provide a set of process indicators that lead to long-term software process improvement
- The only way to know how/where to improve any process is to
 - Measure specific attributes of the process
 - Develop a set of meaningful metrics based on these attributes
 - Use the metrics to provide indicators that will lead to a strategy for improvement

Metrics in the Process Domain (continued)

- We measure the effectiveness of a process by deriving a set of metrics based on outcomes of the process such as
 - Errors uncovered before release of the software
 - Defects delivered to and reported by the end users
 - Work products delivered
 - Human effort expended
 - Calendar time expended
 - Conformance to the schedule
 - Time and effort to complete each generic activity

Etiquette of Process Metrics

- Use common sense and organizational sensitivity when interpreting metrics data
- Provide regular feedback to the individuals and teams who collect measures and metrics
- Don't use metrics to evaluate individuals
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them
- Never use metrics to threaten individuals or teams
- Metrics data that indicate a problem should not be considered “negative”
 - Such data are merely an indicator for process improvement
- Don't obsess on a single metric to the exclusion of other important metrics

Metrics in the Project Domain

Metrics in the Project Domain

- Project metrics enable a software project manager to
 - Assess the status of an ongoing project
 - Track potential risks
 - Uncover problem areas before their status becomes critical
 - Adjust work flow or tasks
 - Evaluate the project team's ability to control quality of software work products
- Many of the same metrics are used in both the process and project domain
- Project metrics are used for making tactical decisions
 - They are used to adapt project workflow and technical activities

Use of Project Metrics

- The first application of project metrics occurs during estimation
 - Metrics from past projects are used as a basis for estimating time and effort
- As a project proceeds, the amount of time and effort expended are compared to original estimates
- As technical work commences, other project metrics become important
 - Production rates are measured (represented in terms of models created, review hours, function points, and delivered source lines of code)
 - Error uncovered during each generic framework activity (i.e, communication, planning, modeling, construction, deployment) are measured

Use of Project Metrics (continued)

- Project metrics are used to
 - Minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks
 - Assess product quality on an ongoing basis and, when necessary, to modify the technical approach to improve quality
- In summary
 - As quality improves, defects are minimized
 - As defects go down, the amount of rework required during the project is also reduced
 - As rework goes down, the overall project cost is reduced

Software Project Planning

The overall goal of project planning is to establish a pragmatic strategy for controlling, tracking, and monitoring a complex technical project.

Why?

So the end result gets done on time, with quality!

Estimation

- Estimation of resources, cost, and schedule for a software engineering effort requires
 - experience
 - access to good historical information (metrics)
 - the courage to commit to quantitative predictions when qualitative information is all that exists
- Estimation carries inherent risk and this risk leads to uncertainty

Estimation Techniques

- Past (similar) project experience
- Conventional estimation techniques
 - task breakdown and effort estimates
 - size (e.g., FP) estimates
- Empirical models
- Automated tools

Conventional Two types of Metrics

- Size oriented
- Function Point oriented

Size-oriented Metrics

- Derived by normalizing quality and/or productivity measures by considering the size of the software produced
- Thousand lines of code (KLOC) are often chosen as the normalization value
- Metrics include
 - Errors per KLOC
 - Defects per KLOC
 - Dollars per KLOC
 - Pages of documentation per KLOC
 - Errors per person-month
 - KLOC per person-month
 - Dollars per page of documentation

Size-oriented Metrics (continued)

- Size-oriented metrics are not universally accepted as the best way to measure the software process
- Opponents argue that KLOC measurements
 - Are dependent on the programming language
 - Penalize well-designed but short programs
 - Cannot easily accommodate nonprocedural languages
 - Require a level of detail that may be difficult to achieve

Example: LOC Approach

Average productivity for systems of this type = 620 LOC/pm.

Burdened labor rate = \$8000 per month, the cost per line of code is approximately \$13.

Based on the LOC estimate and the historical productivity data, the total estimated project cost is **\$431,000 and the estimated effort is 54 person-months.**

An Example of LOC-Based Estimation

Function	Estimated LOC
User interface and control facilities (UCIF)	2300
Two-dimensional geometric analysis	5300
Three-dimensional geometric analysis	6800
Database management	3350
Computer graphics display facilities	4950
Peripheral control function	2100
Design Analysis Modules	8400
<i>Estimated lines of code</i>	<i>33200</i>

Function-Oriented Metrics

- Function-oriented metrics use a measure of the functionality delivered by the application as a normalization value
- Most widely used metric of this type is the function point:

$$FP = \text{count total} * [0.65 + 0.01 * \text{sum (value adj. factors)}]$$

- Function point values on past projects can be used to compute, for example, the average number of lines of code per function point (e.g., 60)

Function Points (5 characteristics)

Based on a combination of program 5 characteristics

The number of :

- External (user) inputs: input transactions that update internal files
- External (user) outputs: reports, error messages
- User interactions: inquiries
- Logical internal files used by the system:



Example a purchase order logical file composed of 2 physical files/tables Purchase_Order and Purchase_Order_Item

- External interfaces: files shared with other systems

Function Points (FP)

- A weight is associated with each of the above 5 characteristics
- Weight range:
 - from 3 for simple feature to
 - 15 for complex feature
- The function point count is computed by multiplying each raw count by the weight and summing all values

FP Calculation

measurement parameter	count		weighting factor				
			simple	avg.	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 ext.interfaces	<input type="text"/>	X	5	7	10	=	<input type="text"/>
count-total							<input type="text"/>
complexity multiplier							<input type="text"/>
function points							<input type="text"/>

Adjusted Function Points Count Complexity: 14 Factors Fi

14 factors: Each factor is rated on a scale of:

Zero: not important or not applicable

Five: absolutely essential

- 1.Backup and recovery
- 2.Data communication
- 3.Distributed processing functions
- 4.Is performance critical?
- 5.Existing operating environment
- 6.On-line data entry
- 7.Input transaction built over multiple screens

Adjusted Function Points Count Complexity:

14 Factors Fi

8.Master files updated on-line

9.Complexity of inputs, outputs, files, inquiries

10.Complexity of processing

11.Code design for re-use

12.Are conversion/installation included in design?

13.Multiple installations

14.Application designed to facilitate change by the user

Final computation

- Value adjustment factor (F_i) = SUM (Backup and recovery +
Data communication +
Performance Criteria etc..)
- $FP_{estimated} = Count_total * [0.65 + 0.01 * \sum (F_i)]$

Function Points

- Each of the F_i criteria are given a rating of 0 to 5 as:
 - No Influence = 0; Incidental = 1;
 - Moderate = 2 Average = 3;
 - Significant = 4 Essential = 5

Function-Oriented Metrics

- Once function points are calculated, they are used in a manner analogous to LOC as a measure of software productivity, quality and other attributes, e.g.:
 - productivity FP/person-month
 - quality faults/FP
 - cost \$\$/FP
 - documentation doc_pages/FP

Example: Function Points

# of user inputs: {on, off, ext. number} (3) x simple (3)	= 9
# of user outputs: {tone} (1) x simple (4)	= 4
# of user inquiries: 0 x simple	= 0
# of files: {mapping table} (1) x simple (7)	= 7
# of external interfaces: {memory map} (1) x simple (5)	= 5
<hr/>	
Total count	= 25

Example 1

- Consider the *average* as the degree of complexity with following functional units and compute the function point for the project
 - ❑ Number of input = 40
 - ❑ Number of output = 28
 - ❑ Number of user enquiries = 32
 - ❑ Number of user files = 9
 - ❑ Number of external interface = 5
- ❑ Adjustment factor = 1.07

Example 2

- i] Number of user inputs=5, with degree of complexity equal to simple
- ii] Number of user output=10, with degree of complexity equal to average
- iii] Number of user enquiries=5, with degree of complexity equal to complex
- iv] Number of user files=8, with degree of complexity equal to simple
- v] Number of external interfaces=3, with degree of complexity equal to complex

- vi] Backup and recovery= 0.5
- vi] Adjustment factor excluding Backup and recovery is 1.07, compute the function point for the project.

Empirical Estimation Method COCOMO II

The COCOMO II Model

- The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model
- Based on the complexity the project can be divided into 3 different modes
 - ❑ Organic mode (simple)
 - ❑ Semi-detached mode (medium)
 - ❑ Embedded mode (complex)
- $\text{Effort} = a * \text{KLOC}^b$, (in person/months)
- $\text{Duration} = c * \text{effort}^d$, (in months)
- $\text{Staffing} = \text{Effort} / \text{Duration}$

The COCOMO II Model

- Organic mode (simple)

$a = 2.4, b = 1.05, c = 2.5, d = 0.38$

- Semi-detached mode (medium)

$a = 3, b = 1.12, c = 2.5, d = 0.35$

- Embedded mode (complex)

$a = 3.6, b = 1.2, c = 2.5, d = 0.32$

Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- per month. Determine the effort required to develop the software product and the nominal development time. From the basic COCOMO estimation formula for organic software:

From the basic COCOMO estimation formula for organic software:

$$\text{Effort} = 2.4 \times (32)^{1.05} = 91 \text{ PM}$$

$$\text{Nominal development time} = 2.5 \times (91)^{0.38} = 14 \text{ months}$$

$$\begin{aligned} \text{Cost required to develop the product} &= 14 \times 15,000 \\ &= \text{Rs. 210,000/-} \end{aligned}$$

Reconciling LOC and FP Metrics

- Relationship between LOC and FP depends upon
 - The programming language that is used to implement the software
 - The quality of the design
- FP and LOC have been found to be relatively accurate predictors of software development effort and cost
 - However, a historical baseline of information must first be established
- LOC and FP can be used to estimate object-oriented software projects
 - However, they do not provide enough granularity for the schedule and effort adjustments required in the iterations of an evolutionary or incremental process
- The table on the next slide provides a rough estimate of the average LOC to one FP in various programming languages

LOC Per Function Point

Language	Average	Median	Low	High
Ada	154	--	104	205
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
COBOL	77	77	14	400
Java	55	53	9	214
PL/1	78	67	22	263
Visual Basic	47	42	16	158

www.qsm.com/?q=resources/function-point-languages-table/index.html

Metrics for Software Quality

- Correctness

- This is the number of defects per KLOC, where a defect is a verified lack of conformance to requirements
- Defects are those problems reported by a program user after the program is released for general use

- Maintainability

- This describes the ease with which a program can be corrected if an error is found, adapted if the environment changes, or enhanced if the customer has changed requirements
- Mean time to change (MTTC) : the time to analyze, design, implement, test, and distribute a change to all users
 - Maintainable programs on average have a lower MTTC

Defect Removal Efficiency

- Defect removal efficiency provides benefits at both the project and process level
- It is a measure of the filtering ability of QA activities as they are applied throughout all process framework activities
 - It indicates the percentage of software errors found before software release
- It is defined as $DRE = E / (E + D)$
 - E is the number of errors found before delivery of the software to the end user
 - D is the number of defects found after delivery
- As D increases, DRE decreases (i.e., becomes a smaller and smaller fraction)
- The ideal value of DRE is 1, which means no defects are found after delivery
- DRE encourages a software team to institute techniques for finding as many errors as possible before delivery