SOFTWARE PROCESS AND PROJECT METRICS

Chapter 4

Software metrics

Measurement can be applied

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

MEASURES , METRICS , AND INDICATORS

a **measure** provides a quantitative indication of the extent, amount, dimensions, capacity, or size of some attribute of a product or process.

a metric as " 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 .

METRICS IN THE PROCESS AND PROJECT DOMAINS

Process indicators

- enable a software engineering organization to gain insight into the efficacy of an existing process (i.e. , the paradigm, software engineering tasks , work products , and milestones) .

- enable managers and practitioners to assess what works and what doesn't. Process metrics are collected across all projects and over long periods of time. Their intent is to provide indicators that lead to long-term software process improvement.
-

Project indicators

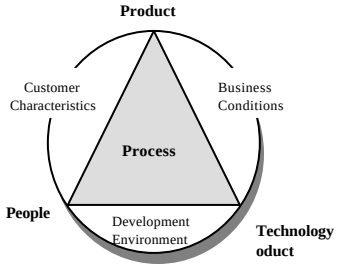
enable a software project manager to

- assess the status of an ongoing project
- track potential risks
- uncover problem areas before they "go critical"
- adjust work flow or tasks
- evaluate the project team's ability to control quality of software engineering work products.

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 and organizational effectiveness.

Determinants for software quality and organizational effectiveness

- **process** sits at the center of a triangle connecting three factors that have a profound influence on software quality and organizational performance
- the skill and motivation of **people** has most influential factor in quality and performance
- the complexity of the **product** have impact on quality and team performance
- the **technology** (the software engineering methods) the process triangle exists within a circle of environmental conditions that include the development environment, business conditions, customer characteristics (e.g., ease of communication)

Outcomes

We measure the efficacy of a software process indirectly based on the outcomes that can be derived from the process.

Outcomes :

- –measures of errors uncovered before release of the software
- –efects delivered to and reported by end users
- –work products delivered
- –human effort expended
- –calendar time expended
- –schedule conformance

We derive process metrics by measuring the characteristics of specific software engineering tasks.

- –measure the effort and time spent performing the umbrella activities
- –measure the generic software engineering activities

Private 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

Examples of metrics private to the individual

- –defect rates (by individual)
- –defect rates (by module)
- –errors found during development

Personal Software Process

A structured set of process descriptions , measurements , and methods that can help engineers to improve their personal performance

Some process metrics are private to the software project team but **public** to all team members

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

Public metrics

Public metrics assimilate information that originally was private to individuals and teams.

Project-level defect

- rates ,
- effort,
- calendar times,
- related data

are collected and evaluated in an attempt to uncover indicators that can improve organizational process performance.

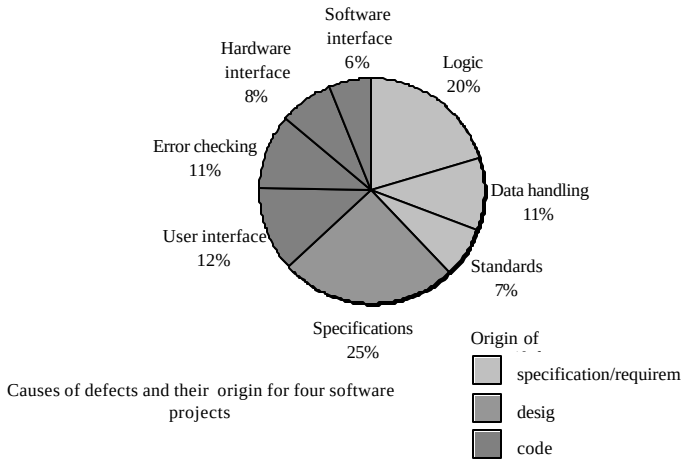
Software metrics etiquette

Software process metrics can be misused.

Software metrics etiquette:

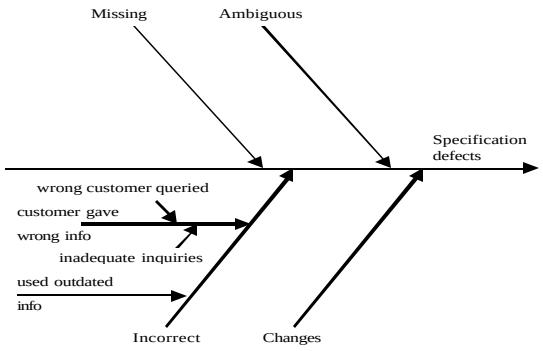
- –Use common sense and organizational sensitivity when interpreting metrics data.
- –Provide regular feedback to the individuals and teams who have worked to collect measures and metries.
- –Don't use metrics to appraise individuals.
- –Work with practitioners and teams to set clear goals and metrics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

A simple defect distribution



Causes of defects and their origin for four software projects

A fishbone diagram showing the causes of one class of defects



Project Metrics

Used for strategic purposes

by a project manager and a software team to adapt project work flow and technical activities .

Project metrics on estimation

effort and time duration estimates

Production rates

–pages of documentation,

–review hours,

–function points,

–delivered source lines

The intent of project metrics is twofold :

–to minimize the development schedule,

–to assess product quality on an ongoing basis and when necessary, modify the technical approach to improve quality.

Every project should measure:

–inputs - measures of the resources (e.g., people, environment) required to do the work,

–outputs - measures of the deliverables or work products created during the software engineering process,

–results - measures that indicate the effectiveness of the deliverables

SOFTWARE MEASUREMENT

Direct measures (e.g., the length of a bolt)
Indirect measures (e.g., the "quality")

Direct measures

–lines of code (LOC)

–execution speed

–memory size

–defects reported over some set period of time

Indirect measures

- functionality, quality, complexity, efficiency, reliability,
- maintainability and many other "abilities"

Size-Oriented Metrics

Derived by normalizing quality and or productivity measures by considering the "size" of the software

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
•	•	•	•	•			
•	•	•	•	•			
•	•	•	•	•			

Size Oriented Metrics

a set of simple size -oriented metrics can be developed for each project:

- errors per KLOC (thousand lines of code)
- defects per KLOC
- $ per LOC
- pages of documentation per YLOC

other interesting metrics:

- –errors/person-month
- –LOC per person-month
- –$/page of documentation

Lines of Code (LOC)

proponents claim

- LOC can be easily counted,
- many existing software estimation models use LOC or KLOC as a key input,
- a large body of literature and data predicated on LOC already exists.

opponents claim

- –LOC measures are programming language dependent,
- –they penalize well-designed but shorter programs,
- –they cannot easily accommodate nonprocedural languages, their use in estimation requires a level of detail that may be difficult to achieve (i.e., the planner must estimate the LOC to be produced long before analysis and design have been completed).

Function-Oriented Metrics

Use a measure of the functionality delivered by the application as a normalization value. Function-oriented metrics were first proposed by Albrecht.

Function points

measurement parameter	count		Weighting Factor: simple	average	complex		
number of user inputs		×	3	4	6	=	
number of user outputs		×	4	5	7	=	
number of user inquiries		×	3	4	6	=	
number of files		×	7	10	15	=	
number of externel interfaces		×	5	7	10	=	
count = total						→	

Computing function point metrics.

Number of user inputs. Each user input that provides distinct applicationoriented data to the software is counted. Inputs should be distinguished from inquiries, which are counted separately.

Number of user outputs. Each user output that provides application-oriented information to the user is counted. In this context output refers to reports, screens, error messages, and so on. Individual data items within a report are not counted separately.

Number of user inquiries. An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.

Number of files. Each logical master file (i.e., a logical grouping of data that may be one part of a large database or a separate file), is counted.

Number of external interfaces. All machine readable interfaces (e.g., data files on tape or disk) that are used to transmit information to another system are counted.

COMPUTING FUNCTION POINTS

Once the above data have been collected, a complexity value is associated with each count.

For determining whether a particular entry,

–simple,

–average,

–complex.

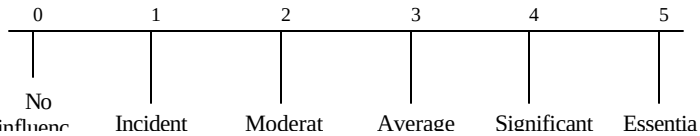
To compute function points (FP),

$$FP = \text{count-total} \times [0.65 + 0.01 \times \Sigma Fi]$$

The *Fi* (i = 1 to 14) are "**complexity adjustment values**"

COMPUTING FUNCTION POINTS

Rate each factor on a scale of 0



1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction 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 designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease o f use by the user?

Function Point

Once function points have been calculated, they are used in a manner analogous to LOC to normalize measures of software productivity, quality, and other attributes:

- errors per FP
- defects per FP
- $ per FP
- page of documentation per FP
- FP per person-month

An Example of LOC-Based Estimation

The CAD software will accept two- and three-dimensional geometric data from an engineer. The engineer will interact and control the CAD system through a user interface that will exhibit characteristics of good humanmachine interface design. All geometric data and other supporting information will be maintained in a CAD database. Design analysis modules will be developed to produce required output which will be displayed on a variety of graphics devices. The software will be designed to control and interact with peripheral devices that include a mouse, digitizer, and laser printer.

- –user interface and control facilities (UICF)
- –two-dimensional geometric analysis (2DGA)
- –three-dimensional geometric analysis (3DGA)
- –database management (DBM)
- –computer graphics display facilities (CGDF)
- –peripheral control (PC)
- –design analysis modules (DAM)

Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control (PC)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	33,200

Estimation table for LOC method
optimistic: 4600
most likely: 6900
pessimistic: 8600

AN EXAMPLE OF FP-BASED ESTIMATION

Information Domain	Value	opt.	likely	pess.	est. count	weight	FP-count
Number of inputs		20	24	30	24	4	96
Number of outputs		12	15	22	16	5	80
Number of inquiries		16	22	28	22	4	88
Number of files		4	4	5	4	10	40
Number of external interfaces		2	2	3	2	7	14
Count-total							**318**

Estimating information domain values.

Factor	Value
Backup and recovery	4
Data communications	2
Distributed processing	0
Performance critical	*4*
Existing operating environment	3
On-line data entry	*4*
Input transaction over multiple screens	*5*
Master files updated on-line	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*
Complexity adjustment factor	1.17

The estimated number of FP is derived:

FPestimated = count-total $\times$ [0.65 + 0.01 $\times$ S *Fi*]

FPestimated = 372

- Historical data normalized using function points indicate that the organizational average productivity for systems of this type is 6.5 FP/pm.
- Burdened labor rate of $8000 per month, the cost per FP is approximately $1230.
- Based on the LOC estimate and the historical productivity data, the total estimated project cost is $457,000 and the estimated effort is 58 person-months.

Extended Function Point Metrics

•The function point metric was originally designed to be applied to business information systems applications.

•The data dimension was emphasized to the exclusion of the functional and behavioral (control) dimensions.

•The function point measure was inadequate for many engineering and embedded systems

•**Feature points**

–a superset of the function point

–applications in which algorithmic complexity is high.

–real-time

–process control

–embedded software applications

To compute the feature point

•information domain values are again counted and weighted as described

•In addition, the feature point metric counts a new software characteristic, algorithms.

•An algorithm is defined as "a bounded computational problem that is included within a specific computer program"

3D Function Point

- **3D function point integrates**
- Data dimension of software with the functional and control dimensions.

Data Dimension

- Counts of retained data (the internal program data structure, e.g., files)
- External data (inputs, outputs, inquiries, and external references)

Functional Dimension

- Measured by considering 'the number of internal operations required to transform input to output data'

Transformation

- A series of processing steps that are constrained by a set of semantic statements.
- A transformation is accomplished with an algorithm that results in a fundamental change to input data as it is processed to become output data.
- Acquire data from a file
- Simply place that data into program memory

The level of complexity assigned to each transformation is a function of :

- The number of processing steps
- The number of semantic statements that control the processing steps.

Processing Steps \ Semantic Statements	1-5	6-10	11+
1-10	low	low	low
11-20	low	average	high
21+	average	high	high

Determining the complexity of a transformation for 3D function points

Control Dimension

- Measured by counting the number of transitions between states
- A state represents some externally observable mode of behavior, and a transition occurs as a result of some event that causes the software or system to change its mode of behavior.

For example, a cellular phone

- **auto-dial** state
- **resting** state
- **dialing** state

To compute 3D function points

$$\text{index} = I + O + Q + F + E + T + R$$

where I, *0, Q, F, E, T,* and R represent inputs, outputs, inquiries, internal data structures, external files, transformations, and transitions, respectively..

- Each complexity weighted value is computed using the following relationship

  complexity weighted value = $N_{il}W_{il} + N_{ia}W_{ia} + N_{ih}W_{ih}$

where N_{il}, N_{ia}, and N_{ih} represent the number of occurrences of element i (e.g., outputs) for each level of complexity (low, average, high)

W_{il}, W_{ia} and W_{ih} are the corresponding weights.

Function Point

- The function point ,like the LOC, measure is controversial

Proponents claim that

- FP is programming language independent, making it ideal for applications using conventional and nonprocedural languages
- It is based on data that are more likely to be known early in the evolution of a project

Opponents claim that

- The method requires some "sleight of hand" in that computation is based on subjective, rather than objective, data
- Counts of the information domain (and other dimensions) can be difficult to collect after-the fact
- FP has no direct physical meaning - it's just a number.

Computing 3-D Function Point

measurement parameter	Complexity Weighting low		average		high		
internal data structures	☐ × 7	+	☐ × 10	+	☐ × 15	=	☐
external data	☐ × 5	+	☐ × 7	+	☐ × 10	=	☐
number of user inputs	☐ × 3	+	☐ × 4	+	☐ × 6	=	☐
number of user outputs	☐ × 4	+	☐ × 5	+	☐ × 7	=	☐
number of user inquiries	☐ × 3	+	☐ × 4	+	☐ × 6	=	☐
transformations	☐ × 7	+	☐ × 10	+	☐ × 15	=	☐
transitions	☐ × n/a	+	☐ × n/a	+	☐ × n/a	=	☐
3D function point index	→						☐

RECONCILING DIFFERENT METRICS APPROACHES

- 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/FP (average)**
assembly language	320
C	128
Cobol	105
Fortran	105
Pascal	90
Ada	70
object-oriented languages	30
fourth generation languages (4GLs)	20
code generators	15
spreadsheets	6
graphical languages (icons)	4

- LOC and FP measures are often used to derive productivity metrics.
- Computing the 3D function point index

Software Productivity

Five important factors that influence software productivity:

- **People factors.** The size and expertise of the development organization.
- **Problem factors**. The complexity of the problem to be solved and the number of changes in design constraints or requirements.
- **Process factors**. Analysis and design techniques that are used, languages and CASE tools available, and review techniques.
- **Product factors.** Reliability and performance of the computer-based system.
- **Resource factors**. Availability of CASE tools and hardware and software resources.

METRICS FOR SOFTWARE QUALITY

The overriding goal of software engineering is to **produce a high-quality system, application, or product**.

The quality of a system, application, or product is only as good as

- The requirements that describe the problem
- The design that models the solution
- The code that leads to an executable program
- The tests that exercise the software to uncover errors.

To accomplish this real-time quality assessment, the engineer must use **technical measures** to evaluate quality in objective, rather than subjective, ways.

Measuring Quality

Correctness :

- The degree to which the software performs its required function.

Common measure:

- **Defects per KLOC**, where a defect is defined as a verified lack of conformance to requirements.

Maintainability:

- The ease with which a program can be corrected if an error is encountered, adapted if its environment changes, or enhanced if the customer desires a change in requirements.

A simple time-oriented metric

- Mean-time-to-change (MTTC), the time it takes to analyze the change request, design an appropriate modification, implement the change, test it, and distribute the change to all users

Measuring Quality

Integrity

- Measures a system's ability to withstand attacks (both accidental and intentional) on its security.
- Attacks programs, data, and documents.
- To measure integrity, two additional attributes must be defined
  - Threat
  - Security

  Threat: The probability that an attack of a specific type will occur within a given time.

  Security The probability that the attack of a specific type will be repelled.

$$\text{integrity} = \Sigma\ [1 - \text{threat} \times (1 - \text{security})]$$

where threat and security are summed over each type of attack.

Measuring Quality

Usability

- User friendliness.
- If a program is not "user friendly," it is often doomed to failure, even if the functions that it performs are valuable
- User friendliness can be measured in terms of four characteristics
  - the physical and/or intellectual skill required to learn the system
  - the time required to become moderately efficient in the use of the system
  - the net increase in productivity measured when the system is used by someone who is moderately efficient
  - a subjective assessment of users attitudes toward the system.

Defect Removal Efficiency

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

where

E = number of errors found before delivery of the software to the end user

D = number of defects found after delivery

- The ideal value for DRE is 1. No defects are found in the software
- Realistically, D will be greater than zero, but the value of DRE can still approach 1 as E increases
- As E increases it is likely that the final value of D will decrease

Defect Removal Efficiency

- DRE can also be used within the project to assess a team's ability to find errors before they are passed to the next framework activity

$$DRE_i = E_i / (E_i + E_{i+1})$$

where

- E_i = number of errors found during software engineering activity i.
- E_{i+1} = number of errors found during software engineering activity i + 1 that are traceable to errors that were not discovered in software engineering activity i.

INTEGRATING METRICS WITHIN THE SOFTWARE PROCESS

- The majority of software developers still do not measure. The problem is cultural. If we do not measure there is no real way of determining whether we are improving.
- Software project managers are concerned with more mundane issues :
  - developing meaningful project estimates
  - producing higher-quality systems
  - getting product out the door on time
- by using measurement to establish a project baseline, each of these issues becomes more manageable.

INTEGRATING METRICS WITHIN THE SOFTWARE PROCESS

As the software design is completed

- Which user requirements are most likely to change?
- Which modules in this system are most error prone?
- How much testing should be planned for each module?
- How many errors (of specific types) can I expect when testing commences?

Answers to these questions can be determined if metrics have been collected and used as a technical guide.

INTEGRATING METRICS WITHIN THE SOFTWARE PROCESS

- **Data collection** requires an historical investigation of past projects to reconstruct required data.
- **Metrics computation** can span a broad range of LOC or FP metrics as well as other quality and project-oriented metrics
- **Metrics evaluation** focuses on the underlying reasons for the results obtained and produces a set of indicators that guide the project or process.

FURTHER READINGS AND OTHER INFORMATION SOURCES

On the Internet,

- Research reports and pointers to information on software metries are available at the Software Engineering Laboratory:

  http://fdd.gsfc.nasa.gov/seltext.html

- The U.S. Army software Metric System Web site contains a variety of useful information on process metrics:

  http://www.army.mil/optecpg/homepage.htm

- A comprehensive listing of textbooks and papers on process metrics can be obtained at:

  http://www.rai.com/soft_eng/sme.html

FURTHER READINGS AND OTHER INFORMATION SOURCES

- A Listserv mailing list that addresses function point metrics has been established. To subscribe, send mail to: **cim@crim.ca**

  SUBJECT : "none" (this field must be empty)

  CONTENT: SUB FUNCTION.POINT.LIST "Your name"

- An up-to-date list of World Wide Web references for software process metrics can be found at:

  http://www.rspa.com