NOTES SOFTWARE PROCESS MEASUREMENT:

Measures, Metrics and Indicators

- A *measure* provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process.
  - E.g., **Number of errors**
- The IEEE glossary defines a *metric* as "a quantitative measure of the degree to which a system, component, or process possesses a given attribute."
  - E.g., **Number of errors found per person hours expended**
- An *indicator* is a metric or combination of metrics that provide insight into the software process, a software project, or the product itself.

Motivation for Metrics

- Estimate the cost & schedule of future projects
- Evaluate the productivity impacts of new tools and techniques
- Establish productivity trends over time
- Improve software quality
- Forecast future staffing needs
- Anticipate and reduce future maintenance needs

Project Metrics

- Used during estimation
- Used to monitor and control progress
- The intent is twofold
  - Minimize the development schedule
  - Assess product quality on an ongoing basis
- Leads to a reduction in overall project cost

Software Measurement

- S/W measurement can be categorized in two ways:
  1. ***Direct measures*** of the s/w process (e.g., cost and effort applied) and product (e.g., lines of code (LOC) produced, etc.)
  2. ***Indirect measures*** of the product (e.g., functionality, quality, complexity, etc.)
- Requires normalization of both size- and function-oriented metrics

Size-Oriented Metrics (1)

- Lines of Code (LOC) can be chosen as the normalization value
- Example of simple size-oriented metrics
  - Errors per KLOC (thousand lines of code)
  - Defects per KLOC
  - $ per KLOC
  - Pages of documentation per KLOC

Size-Oriented Metrics (2)

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 (3)

- Controversy regarding use of LOC as a key measure
  - According to the proponents
    - LOC is an "artifact" of all s/w development projects
    - Many existing s/w estimation models use LOC or KLOC as a key input
  - According to the opponents
    - LOC measures are programming language dependent
    - They penalize well-designed but shorter programs
    - Cannot easily accommodate nonprocedural languages
    - Difficult to predict during estimation

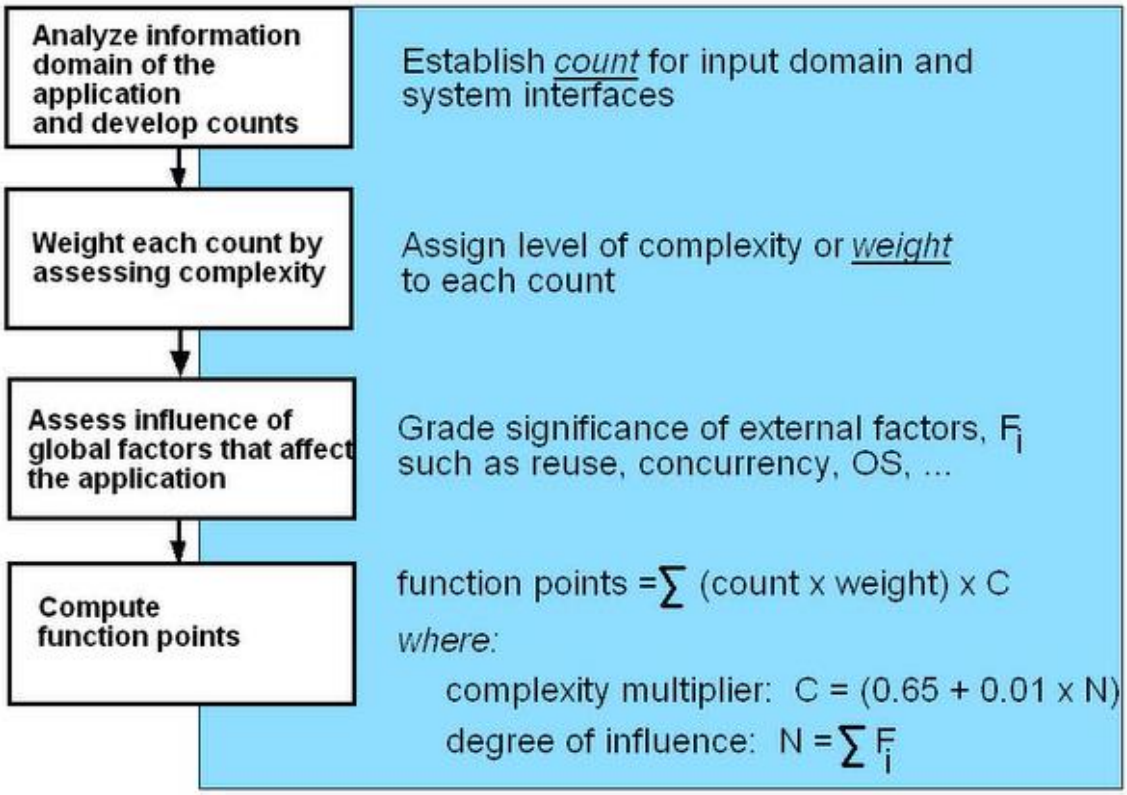
Function-Oriented Metrics

- The most widely used function-oriented metric is the ***function point*** (FP)
- Computation of the FP is based on characteristics of the software's information domain and complexity

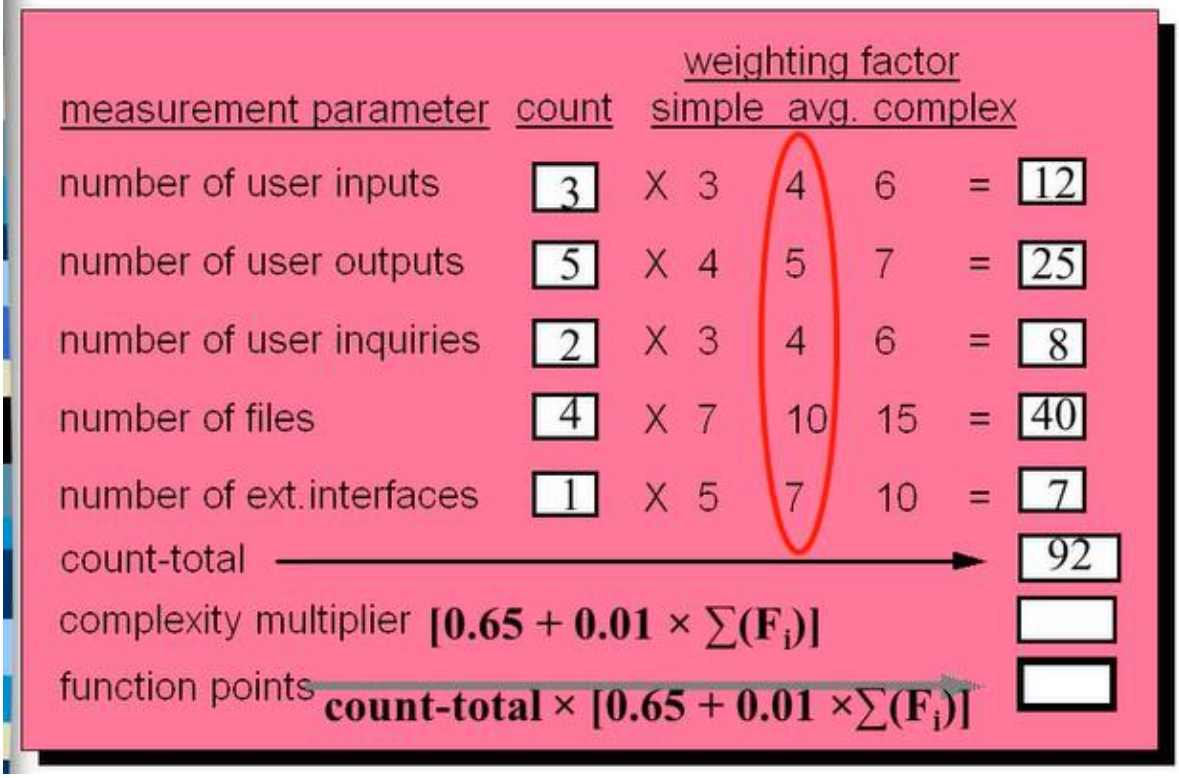
Information Domain

- **Number of external inputs** – from user or another application
- **Number of external outputs**
- **Number of external inquiries** – request from user that generates an on-line output
- **Number of internal logical files** (maintained by system)
- **Number of external interface files** (provides data but not maintained by system)

Computing Function Points



Analyzing the Information Domain



measurement parameter	count	weighting factor simple	avg.	complex		
number of user inputs	3	X 3	4	6	=	12
number of user outputs	5	X 4	5	7	=	25
number of user inquiries	2	X 3	4	6	=	8
number of files	4	X 7	10	15	=	40
number of ext.interfaces	1	X 5	7	10	=	7
count-total →						92
complexity multiplier $[0.65 + 0.01 \times \sum(F_i)]$						
function points **count-total** $\times [0.65 + 0.01 \times \sum(F_i)]$						

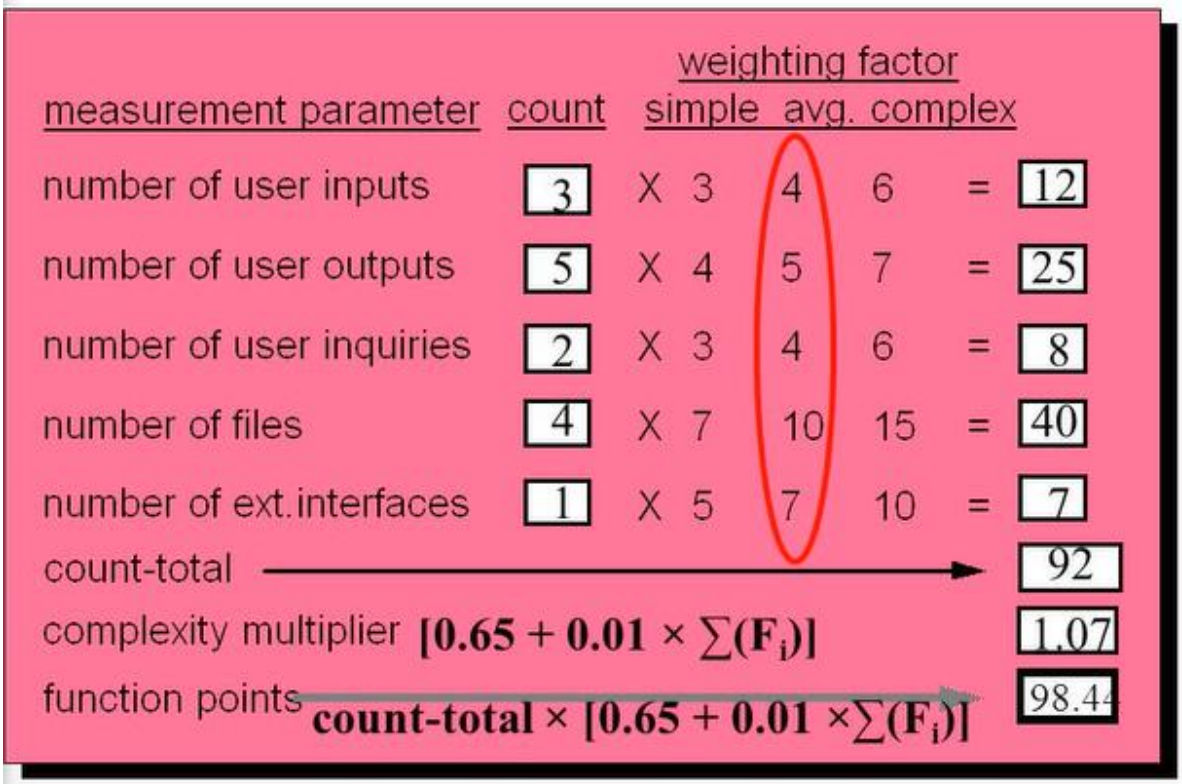
Taking Complexity into Account

Factors(F_i) are rated on a scale of 0 (not important) to 5 (essential)

The following are some examples of these factors:

- Is high performance critical?
- Is the internal processing complex?
- Is the system to be used in multiple sites and/or by multiple organizations?
- Is the code designed to be reusable?
- Is the processing to be distributed?
- and so forth . . .

Computing Function Points



Uses of Function Points(FP)

But how long will the project take and how much will it cost?

- If programmers in an organization produce average 16 function points per month. Thus . . .

  98.44 FP divided by 16 = 6 man-months

- If the average programmer is paid $5,200 per month (including benefits), then the [labor] cost of the project will be . . .

  6 man-months X $5,200 = $31,200

Pros & Cons of FP

- Controversy regarding use of FP as a key measure
  - According to the proponents
    - It is programming language independent
    - Can be predicted before coding is started
  - According to the opponents
    - Based on ***subjective*** rather than ***objective*** data
    - Has no direct physical meaning – it's just a number

Reconciling LOC and FP Metrics

Programming Language	LOC per Function point			
	Avg.	Median	Low	High
Access	35	38	15	47
Ada	154	—	104	205
APS	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
Cool:Gen/IEF	38	31	10	180
Culprit	51	—	—	—
DBase IV	52	—	—	—
Easytrieve+	33	34	25	41
Excel47	46	—	31	63
Focus	43	42	32	56
FORTRAN	—	—	—	—
FoxPro	32	35	25	35
Ideal	66	52	34	203
IEF/Cool:Gen	38	31	10	180
Informix	42	31	24	57
Java	63	53	77	

Key Project & Process Metric Groups

Project managers have a wide variety of metrics to choose from. We can classify the most commonly used metrics into the following groups:

1. Process Metrics

   These are metrics that pertain to Process Quality. They are used to measure the efficiency and effectiveness of various processes.

2. Project Metrics

   These are metrics that relate to Project Quality. They are used to quantify defects, cost, schedule, productivity and estimation of various project resources and deliverables.

3. Product Metrics

   These are metrics that pertain to Product Quality. They are used to measure cost, quality, and the product's time-to-market.

4. Organizational Metrics

   These metrics measure the impact of organizational economics, employee satisfaction, communication, and organizational growth factors of the project.

5. Software Development Metrics Examples

   These metrics enable management to understand the quality of the software, the productivity of the development team, code complexity, customer satisfaction, agile process, and operational metrics.

We'll now take a closer look at the various types of the two most important categories of metrics – Project Metrics, and Process Metrics.

Project Metrics

1. Schedule Variance: Any difference between the scheduled completion of an activity and the actual completion is known as Schedule Variance.

   Schedule variance = ((Actual calendar days – Planned calendar days) + Start variance)/ Planned calendar days x 100.

2. Effort Variance: Difference between the planned outlined effort and the effort required to actually undertake the task is called Effort variance.

   Effort variance = (Actual Effort – Planned Effort)/ Planned Effort x 100.

3. Size Variance: Difference between the estimated size of the project and the actual size of the project (normally in KLOC or FP).

   Size variance = (Actual size – Estimated size)/ Estimated size x 100.

4. Requirement Stability Index: Provides visibility to the magnitude and impact of requirements changes.

   RSI = 1- ((Number of changed + Number of deleted + Number of added) / Total number of initial requirements) x100.

5. Productivity (Project): Is a measure of output from a related process for a unit of input.

   Project Productivity = Actual Project Size / Actual effort expended in the project.

6. Productivity (for test case preparation) = Actual number of test cases/ Actual effort expended in test case preparation.

7. Productivity (for test case execution) = Actual number of test cases / actual effort expended in testing.

8. Productivity (defect detection) = Actual number of defects (review + testing) / actual effort spent on (review + testing).

9. Productivity (defect fixation) = actual no of defects fixed/ actual effort spent on defect fixation.

10. Schedule variance for a phase: The deviation between planned and actual schedules for the phases within a project.

Schedule variance for a phase = (Actual Calendar days for a phase – Planned calendar days for a phase + Start variance for a phase)/ (Planned calendar days for a phase) x 100

11. Effort variance for a phase: The deviation between a planned and actual effort for various phases within the project.

    Effort variance for a phase = (Actual effort for a phase – a planned effort for a phase)/ (planned effort for a phase) x 100.

Process Metrics:

1. Cost of quality: It is a measure of the performance of quality initiatives in an organization. It's expressed in monetary terms.

   Cost of quality = (review + testing + verification review + verification testing + QA + configuration management + measurement + training + rework review + rework testing)/ total effort x 100.

2. Cost of poor quality: It is the cost of implementing imperfect processes and products.

   Cost of poor quality = rework effort/ total effort x 100.

3. Defect density: It is the number of defects detected in the software during development divided by the size of the software (typically in KLOC or FP)

   Defect density for a project = Total number of defects/ project size in KLOC or FP

4. Review efficiency: defined as the efficiency in harnessing/ detecting review defects in the verification stage.

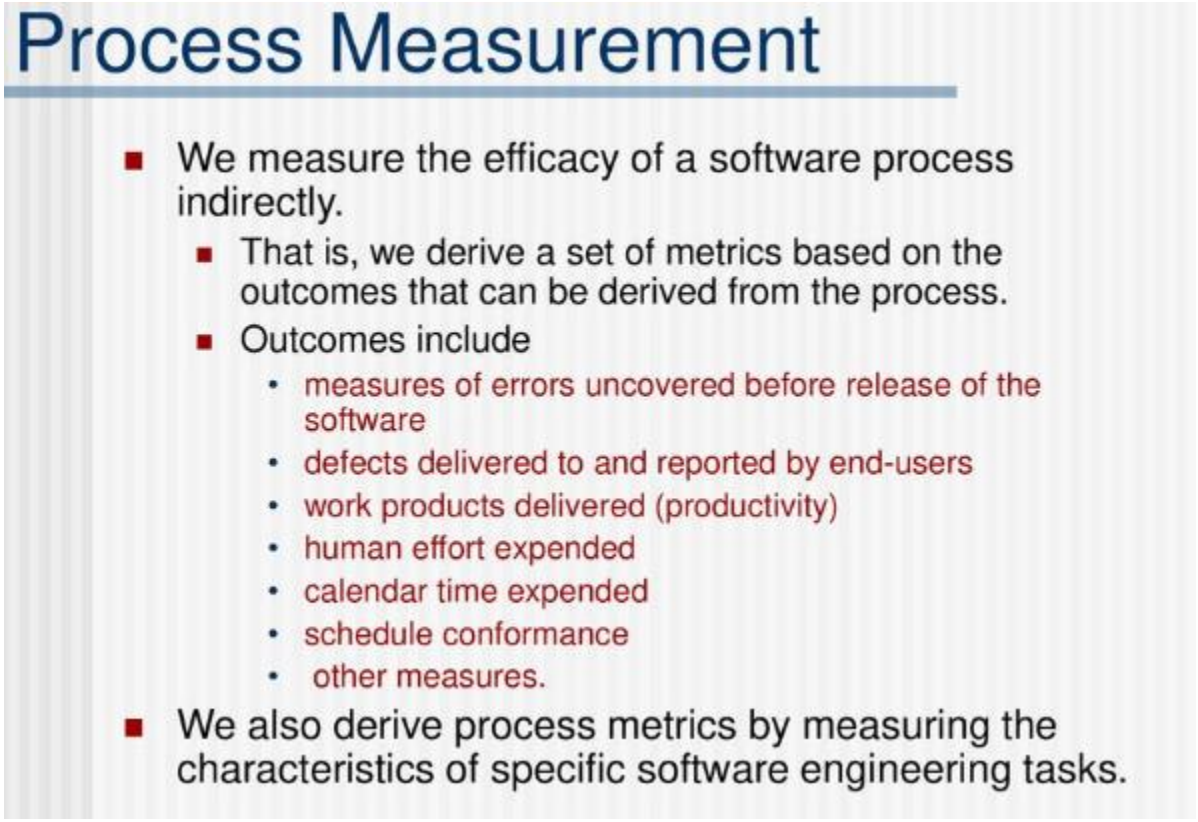
   Review efficiency = (number of defects caught in review)/ total number of defects caught) x 100.

5. Testing Efficiency: Testing efficiency = 1 – ((defects found in acceptance)/ total number of testing defects) x 100.

6. Defect removal efficiency: Quantifies the efficiency with which defects were detected and prevented from reaching the customer.

   Defect removal efficiency = (1 – (total defects caught by customer/ total number of defects)) x 100.

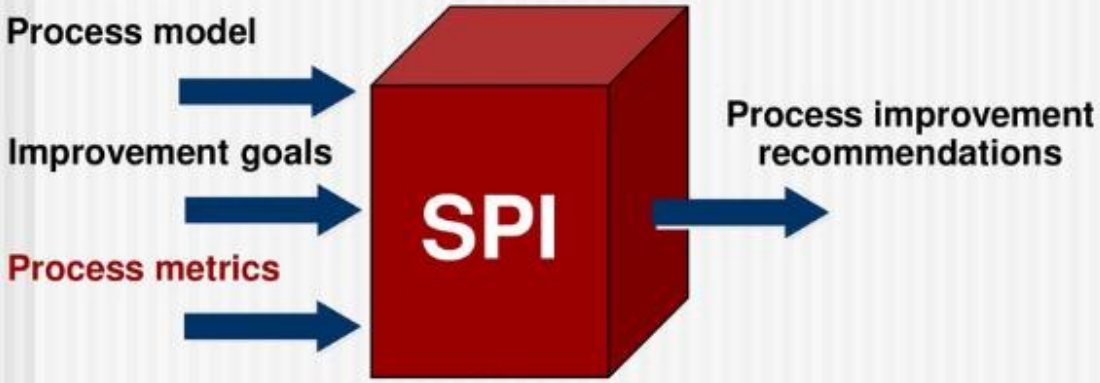
7. Residual defect density = (total number of defects found by a customer)/ (Total number of defects including customer found defects) x 100.



Process Metrics Guidelines

- 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 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.

Software Process Improvement



Project Metrics

- used to minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks
- used 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, tools) 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.

Typical Project Metrics

- Effort/time per software engineering task
- Errors uncovered per review hour
- Scheduled vs. actual milestone dates
- Changes (number) and their characteristics
- Distribution of effort on software engineering tasks

Metrics Guidelines

- 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 metrics.
- 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.

Typical Size-Oriented Metrics

- errors per KLOC (thousand lines of code)
- defects per KLOC
- $ per LOC
- pages of documentation per KLOC
- errors per person-month
- errors per review hour
- LOC per person-month
- $ per page of documentation

Typical Function-Oriented Metrics

- errors per FP (thousand lines of code)
- defects per FP
- $ per FP
- pages of documentation per FP
- FP per person-month

Comparing LOC and FP

Programming Language	LOC per Function point avg.	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	63	53	77	-
JavaScript	58	63	42	75
Perl	60	-	-	-
PL/1	78	67	22	263
Powerbuilder	32	31	11	105
SAS	40	41	33	49
Smalltalk	26	19	10	55
SQL	40	37	7	110
Visual Basic	47	42	16	158

Representative values developed by QSM

Why Opt for FP?

- Programming language independent
- Used readily countable characteristics that are determined early in the software process
- Does not "penalize" inventive (short) implementations that use fewer LOC that other more clumsy versions
- Makes it easier to measure the impact of reusable components

Object-Oriented Metrics

- Number of scenario scripts (use-cases)
- Number of support classes (required to implement the system but are not immediately related to the problem domain)
- Average number of support classes per key class (analysis class)
- Number of subsystems (an aggregation of classes that support a function that is visible to the end-user of a system)

WebApp Project Metrics

- Number of static Web pages (the end-user has no control over the content displayed on the page)
- Number of dynamic Web pages (end-user actions result in customized content displayed on the page)
- Number of internal page links (internal page links are pointers that provide a hyperlink to some other Web page within the WebApp)
- Number of persistent data objects
- Number of external systems interfaced
- Number of static content objects
- Number of dynamic content objects
- Number of executable functions

Measuring Quality

- Correctness — the degree to which a program operates according to specification
- Maintainability—the degree to which a program is amenable to change
- Integrity—the degree to which a program is impervious to outside attack
- Usability—the degree to which a program is easy to use

Defect Removal Efficiency

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

where:

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.

Metrics for Small Organizations

- time (hours or days) elapsed from the time a request is made until evaluation is complete, t_{queue}.
- effort (person-hours) to perform the evaluation, W_{eval}.
- time (hours or days) elapsed from completion of evaluation to assignment of change order to personnel, t_{eval}.
- effort (person-hours) required to make the change, W_{change}.
- time required (hours or days) to make the change, t_{change}.
- errors uncovered during work to make change, E_{change}.
- defects uncovered after change is released to the customer base, D_{change}.

Establishing a Metrics Program

- Identify your business goals.
- Identify what you want to know or learn.
- Identify your subgoals.
- Identify the entities and attributes related to your subgoals.
- Formalize your measurement goals.
- Identify quantifiable questions and the related indicators that you will use to help you achieve your measurement goals.
- Identify the data elements that you will collect to construct the indicators that help answer your questions.
- Define the measures to be used, and make these definitions operational.
- Identify the actions that you will take to implement the measures.
- Prepare a plan for implementing the measures.

The Measurement Process

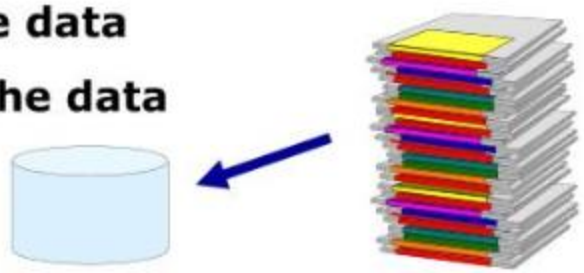
Plan

- **Define goals & information needs**
  - Work with customer & stakeholders
  - Prioritize and align
- **Select measures**
  - See prior module on selecting SW measures
- **Plan the data collection, analysis and reporting procedures**
- **Define criteria for evaluation**
- **Obtain approval for resources**
- **Deploy supporting technologies**

The Measurement Process

Collect

- **Integrate the data collection process into the software & management processes**
  - Consider the need for culture change
  - Consider the impact of data collection
- **Communicate the process to affected parties**
- **Deploy the collection process**
- **Collect & verify the data**
- **Record and store the data**



The Measurement Process

Analyze

- **Analyze the Data**
  - Interpret relative to models, etc.
- **Report/Communicate the Results**
  - Document
  - Display/Communicate to users
  - Interpret

The Measurement Process

Utilize

- **Change the project**
  - Identify trends early
  - Adjust plans to head off problems
- **Change the process**
  - At the project level
  - At the organizational level
- **Change the measurement process**
  - Data definitions, collection process, validation, etc.

Evaluate Measurements (not part of core process)

Example: Same Base Measure, Different Uses

Base Measure: Number of defects in the software

Use in a Project Measure: **number of defects must be less than target before project is complete, thus the data are used to measure project status.**

Use in a Product Measure: **product quality index is "defects per 1000 LOC". Used to determine probable warranty cost.**

Use in a Process Measure: **quality index is compared for different processes and for process improvements to determine which processes are best for future projects in the organization**

Example

- ***Project measure*****: schedule performance**
- ***Problem:*** **schedule performance is poor (we are behind schedule)**
- ***Solution*****: understand why the process is not achieving the desired schedule**
  - Are we not following the process?
  - Is the process inefficient?
  - Are people poorly trained?
  - Is the process ill suited to this project?

Questions to help identify process measures

Example: A *Measure & its Impact*

Information Need: Productivity

Measure 1: Lines of code per day

Use: reward those who produce the most lines of code per day

Result: people produce bloated, inflated code in order to look good

---

Measure 2: Requirements met and tested, weighted by complexity of requirement

Use: track against history and use to identify process bottlenecks

Result: people will use the data to make the process more efficient, resulting in lower cost

Good and Not-So-Good Measures

Goal: *Produce software more efficiently*
Information Needed: *Efficiency*

Measure 1: tests completed per week
Result: easy tests done first; corners cut in testing; hard problems ignored or deferred

Measure 2: rework
Result: process and methods are improved to reduce rework, resulting in more efficient software development

- But rework is a lagging indicator - it does not spot problems in advance

What Attributes Can We Measure?

- **We want attributes that relate to our goals**
  - time, resources, performance, quality etc.
- **The following type of matrix can help:**

What / Attributes	Process	Product	Project
Time	What Is our Cycle Time?	How Fast can we Manufacture?	Are We On Schedule?
Resources	What is our Productivity?	What will it Cost?	Expenses vs. Budget?
Performance	Does it Work?	Meets Perf. Goals?	Meets Mgt. Goals?
Quality	In-process Defects?	Post-release Defects?	Customer Satisfaction?

Unit 11: Software Metrics

Objective

- To describe the current state-of-the-art in the measurement of software products and process.

Why Measure?

- "When you can measure what you are speaking about and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind: it may be the beginnings of knowledge but you have scarcely in your thoughts advanced to the stage of Science."
  - Lord Kelvin (Physicist)
- "You cannot control what you cannot measure."
  - Tom DeMarco (Software Engineer)

What is Measurement

- measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined unambiguous rules

Examples of Entities and Attributes

Entity	*Attribute*
Software Design	Defects discovered in design reviews
Software Design Specification	Number of pages
Software Code	Number of lines of code, number of operations
Software Development Team	Team size, average team experience

Types of Metric

- direct measurement
  - eg number of lines of code
- indirect/ derived measurement
  - eg defect density = no. of defects in a software product / total size of product
- prediction
  - eg predict effort required to develop software from measure of the functionality – function point count

Types of Metric

- nominal
  - eg no ordering, simply attachment of labels
    (language: 3GL, 4GL)
- ordinal
  - eg ordering, but no quantitative comparison
    (programmer capability: low, average, high)

Types of Metric

- interval
  - eg between certain values
    (programmer capability: between 55th and 75th percentile of the population ability)
- ratio
  - eg (the proposed software is twice as big as the software that has just been completed)
- absolute
  - eg the software is 350,000 lines of code long

Project Metrics

- used to minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks
- used 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, tools) 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.

Web Engineering Project Metrics (2)

- Let,
  - N_{sp} = number of static Web pages
  - N_{dp} = number of dynamic Web pages
- Then,
  - Customization index, $C = N_{dp}/(N_{dp} + N_{sp})$
- The value of C ranges from 0 to 1

Metrics for Software Quality

- Goals of s/w engineering
  - Produce high-quality systems
  - Meet deadlines
  - Satisfy market need

- The primary thrust at the project level is to measure errors and defects

Measuring Quality

- Correctness
  - Defects per KLOC
- Maintainability
  - Mean-time-to-change (MTTC)
- Integrity
  - Threat and security
  - integrity = Σ [1 – (threat $\times$ (1 - security))]
- Usability

Defect Removal Efficiency (DRE)

- Can be used at both the project and process level
- DRE = E / (E + D), [E = Error, D = Defect]
- Or, $DRE_i = E_i / (E_i + E_{i+1})$, [for i^{th} activity]
- Try to achieve DRE_i that approaches 1