# SOFTWARE PROJECT MANAGENT

Unit 3

# Unit 3 - Syllabus

- **Technical Metrics For Software-Software** Process and Project Metrics- Size Oriented Metrics- Function-Oriented Metrics- Extended Function Point Metrics. A Framework for Technical Software Metrics- Metrics for Requirement Specification Quality- **Metrics for Analysis-** Metrics for Design- Metrics for Source Code- Metrics for Testing- Metrics for Maintenance.

  *(Chapter 4 – **Software Engineering , Roger S Pressman**)*

- **Technical Metrics For Object-Oriented Systems-Intent** of Object-Oriented Metrics- Characteristics of Object-Oriented Metrics - Metrics for OO Design Model- Class-Oriented Metrics- Operation-Oriented Metrics- Metrics for Object-Oriented Testing- Metrics for Object-Oriented Projects.

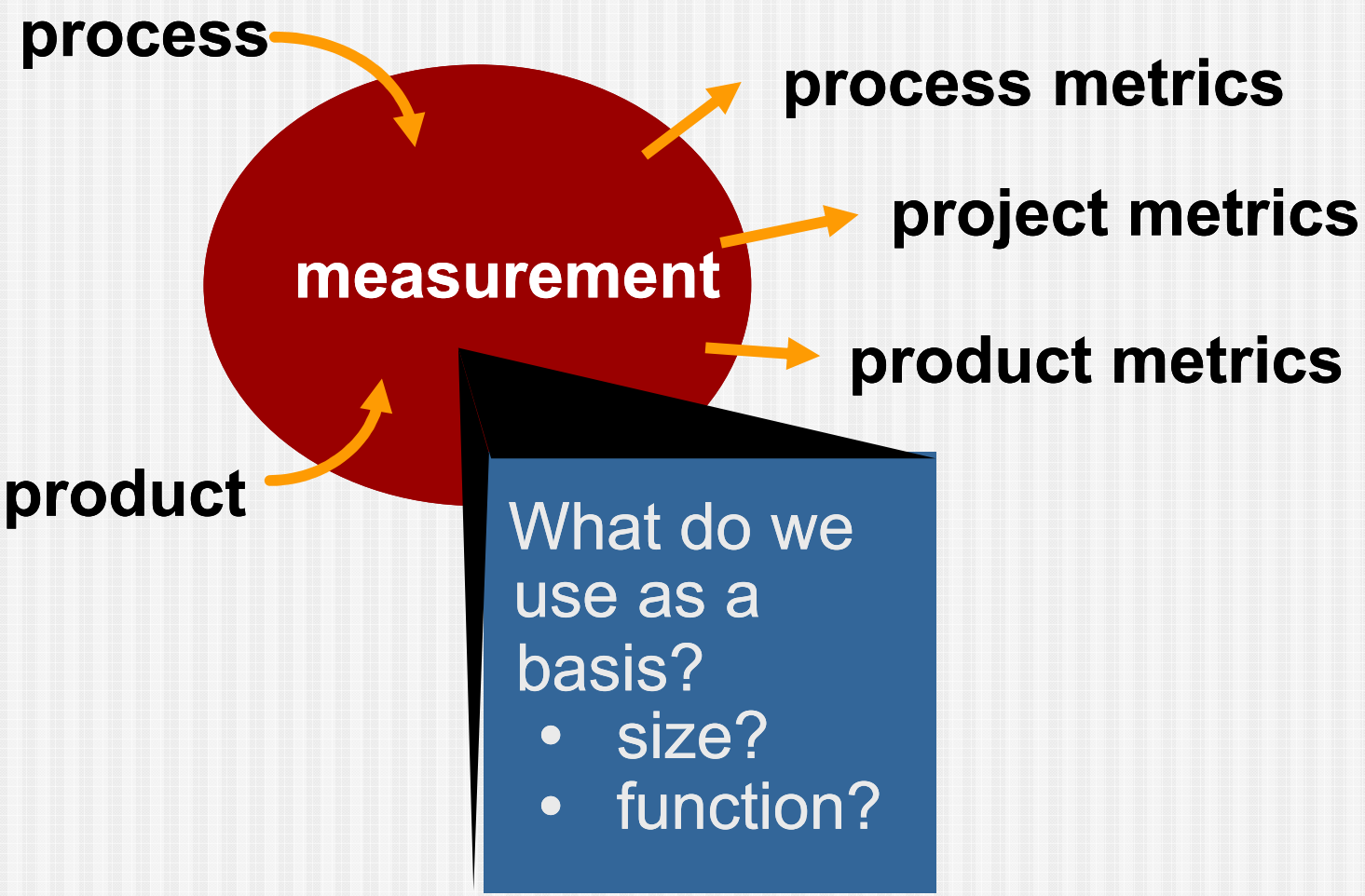
  *(Chapter 24 – Software Engineering , Roger S Pressman)*

# Software Metrics - Definition

- A software metric is a standard of measure of a degree to which a software system or process possesses some property.
- The goal is obtaining quantitative measurements to objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance, testing, software debugging, software performance optimization, and optimal personnel task assignments.

*Reference : https://en.wikipedia.org/wiki/Software_metric*

# A Good Manager Measures



# Why Do We Measure?

- 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 work products.

# Metrics

- **Metrics** is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.
- **Measures, Metrics, and Indicators :** An **indicator** is a metric or combination of metrics that provide insight into the software process, a software project, or the product itself.

# Need for Software Metrics

- **in order to**
  - Gain an understanding of processes, products, resources, and environments.
  - Establish baselines for comparisons with future assessments
- To **evaluate in order to :**
  - Determine status with respect to plans
- To **predict in order to**
  - Gain understanding of relationships among processes and products.
  - Build models of these relationships
- To **improve in order to**
  - Identify roadblocks, root causes, inefficiencies, and other opportunities for improving product quality and process performance.

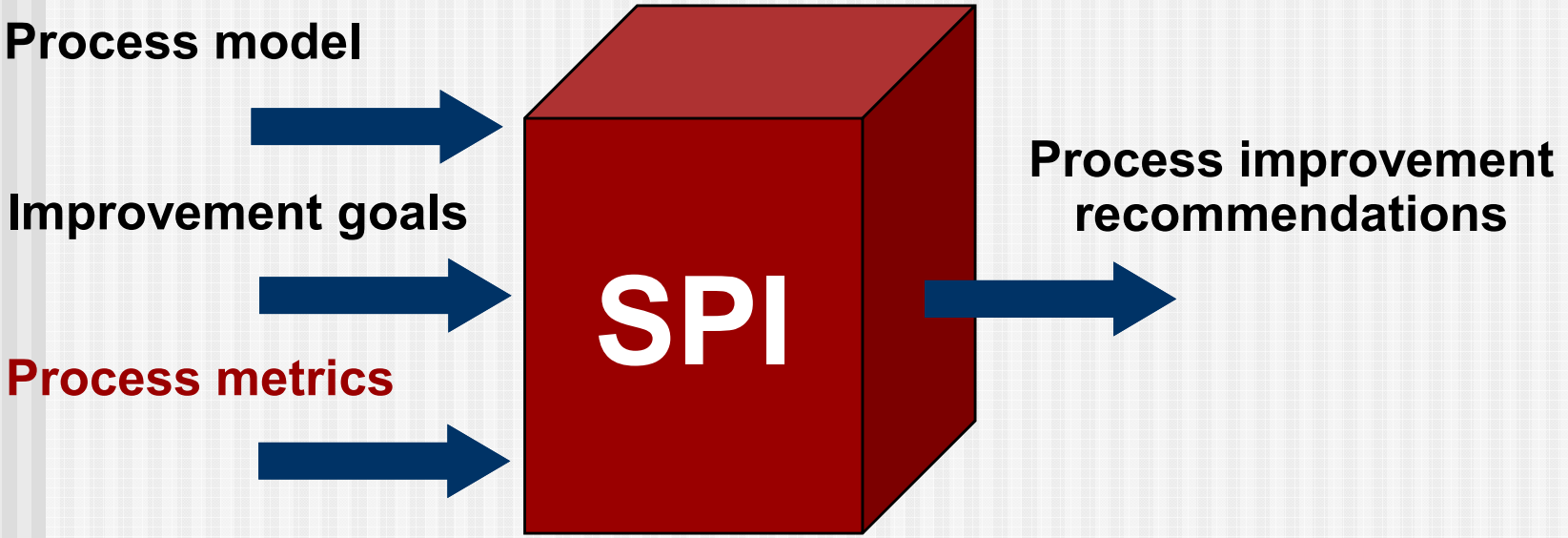
# Process Measurement

- We measure the efficacy of a software process indirectly.
  - That is, we derive a set of metrics based on the outcomes that can be derived from the process.
  - Outcomes include
    - measures of errors uncovered before release of the software
    - defects delivered to and reported by end-users
    - work products delivered (productivity)
    - human effort expended
    - calendar time expended
    - schedule conformance
    - other measures.
- We also derive process metrics by measuring the characteristics of specific software engineering tasks.

# 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



# Process Metrics

- Quality-related
  - focus on quality of work products and deliverables
- Productivity-related
  - Production of work-products related to effort expended
- Statistical SQA data
  - error categorization & analysis
- Defect removal efficiency
  - propagation of errors from process activity to activity
- Reuse data
  - The number of components produced and their degree of reusability

# 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.

# Types of Metrics

- Size Oriented Metrics
- Function Oriented Metrics
- Object Oriented Metrics

# Size Oriented Metrics

- Size Oriented Metrics derived by normalizing quality and productivity Point Metrics measures by considering size of the software that has been produced. The organization builds a simple record of size measure for the software projects. It is built on past experiences of organizations. It is a direct measure of software.
- This metrics is one of simplest and earliest metrics that is used for computer program to measure size. Size Oriented Metrics are also used for measuring and comparing productivity of programmers. It is a direct measure of a Software. The size measurement is based on lines of code computation. The lines of code are defined as one line of text in a source file.

# 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.

# OBJECT ORIENTED METRICS

# 3.1 Introduction to OO Metrics

- OO metrics have been introduced to help a software engineer use quantitative analysis to assess the quality of the design before a system is built.
- The focus of OO metrics is on the class—the fundamental building block of the OO architecture.
- Software engineers use OO metrics to help them build higher-quality software.

# 3.2 Technical Metrics For Object-Oriented Systems

- **Goals for Using Object-Oriented Metrics**
  - To better understand product quality
  - To assess process effectiveness
  - To improve quality of the work performed at the project level

# 3.2.1 Steps in Measuring OO

- **Step 1:** The measurement process is to derive the software measures and metrics that are appropriate for the representation of software that is being considered.
- **Step 2:** Once computed, appropriate metrics are analyzed based on pre-established guidelines and past data.
- **Step 3 :** The results of the analysis are interpreted to gain insight into the quality of the software, and the results of the interpretation lead to modification of work products arising out of analysis, design, code, or test.

# 3.3 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)

# 3.4 Characteristics of OO Metrics

- **Localization** - OO metrics need to apply to the class as a whole and should reflect the manner in which classes collaborate with one another
- **Encapsulation** - OO metrics chosen need to reflect the fact that class responsibilities, attributes, and operations are bound as a single unit
- **Information hiding** -OO metrics should provide an indication of the degree to which information hiding has been achieved
- **Inheritance** - OO metrics should reflect the degree to which reuse of existing classes has been achieved (number of *children* , number of *parents* , and class hierarchy *nesting level)*
- **Abstraction** - OO metrics represent abstractions in terms of measures of a class (e.g. number of instances per class per application)

# 3.5 OO Design Model Metrics

- Size(population, volume, length, functionality)
- Complexity (how classes interrelate to one another)
- Coupling (physical connections between design elements)
- Sufficiency (how well design components reflect all properties of the problem domain)
- Completeness (coverage of all parts of problem domain)
- Cohesion (manner in which all operations work together)
- Primitiveness (degree to which attributes and operations are atomic)
- Similarity (degree to which two or more classes are alike)
- Volatility (likelihood a design component will change)

# 3.6 Class Oriented Metrics

**Class** is often the "parent" for subclasses (sometimes called *children) that inherit its* attributes and operations.

**Measures and metrics** for an individual class, the class hierarchy, and class collaborations are required to design quality.

**Class Oriented Metrics Suites**

1. Chidamber and Kemerer (CK) Metrics Suite
2. Metrics Proposed by Lorenz and Kidd
3. The MOOD Metrics Suite

# 3.6.1 Chidamber and Kemerer (CK) Metrics Suite

- Most widely referenced sets of OO software metrics has been proposed by Chidamber and Kemerer often referred as the *CK metrics suite.*
- CK Metrics : Six class-based design metrics.
    1. Weighted metrics per class (WMC)
    2. Depth of inheritance tree (DIT) - the maximum length from the node to the root of the tree
    3. number of children (NOC)
    4. coupling between object classes (CBO)
    5. response for a class(RFC) - a set of methods that can potentially be executed in response to a message received by an object of that class
    6. lack of cohesion in methods (LCOM)

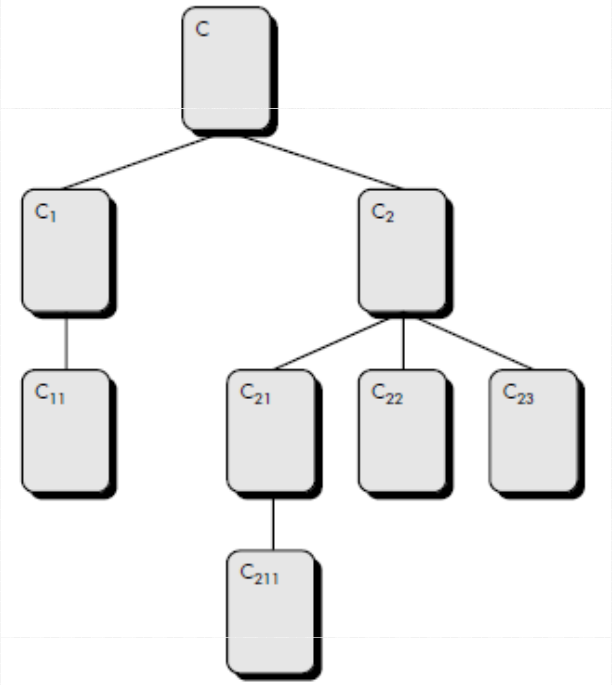
# Weighted methods per class

**Weighted methods per class (WMC).** Assume that $n$ methods of complexity $c_1$, $c_2, \ldots, c_n$ are defined for a class **C**. The specific complexity metric that is chosen (e.g., cyclomatic complexity) should be normalized so that nominal complexity for a method takes on a value of 1.0.

$$\text{WMC} = \Sigma\, c_i$$

*for i = 1 to n.*
*The number of methods and their complexity are reasonable indicators*
*of the amount of effort required to implement and test a class.*

# Number of classess




**C2 has three children—subclasses C21, C22, and C23.**
As the number of children grows, reuse increases but also, as NOC increases, the abstraction represented by the parent class can be diluted.

# 3.6.2 Lorenz and Kidd Metrics

Class-based metrics into four broad categories: size, inheritance, internals, and externals.

1. Class size (CS)
   - The total number of operations (both inherited and private instance operations) that are encapsulated within class.
   - The number of attributes (both inherited and private instance attributes) that are encapsulated by the class.
2. number of operations overridden by a subclass (NOO)
3. number of operations added by a subclass (NOA)
4. specialization index (SI)
   - indication of the degree of specialization for each of the subclasses in an OO system

# 3.6.3 MOOD Metrics

- Harrison, Counsell, and Nithi proposed a set of metrics for object-oriented design that provide quantitative indicators for OO design characteristics.
  1. Method Inheritance Factor (MIF)
     - degree to which the class architecture of an OO system makes use of inheritance for both methods (operations) and attributes
  2. Coupling Factor (CF)
  3. Polymorphism Factor (PF)
     - The number of methods that redefine inherited methods, divided by the maximum number of possible distinct polymorphic situations

# 3.7 Operation Oriented Metrics

- **Operation Oriented Metrics** : Metrics for operations that reside within a class.

1. Average operation size ($OS_{avg}$)
   - the number of messages sent by the operation provides an alternative for operation size
2. Operation complexity (OC)
3. Average number of parameters per operation ($NP_{avg}$)
   - The larger the number of operation parameters, the more complex the collaboration between objects.

# 3.8 Object Oriented Testing Metrics

- Binder suggests a broad array of design metrics that have a direct influence on the “testability” of an OO system.

1. Encapsulation
   - lack of cohesion in methods (LCOM)
   - percent public and protected (PAP)
   - public access to data members(PAD)
2. Inheritance
   - number of root classes (NOR)
   - fan in (FIN)
   - number of children (NOC) and depth of inheritance tree (DIT)

# 3.9 Metrics for OO Projects

- A software team can use software project metrics to adapt project workflow and technical activities.
- Project metrics are used to avoid development schedule delays, to mitigate potential risks, and to assess product quality on an on-going basis.
- Size is directly proportional to effort and duration. The following OO metrics can provide insight into software size:
    1. **Number of scenario scripts (NSS).**
    2. **Number of key classes (NKC).**
    3. (**Number of subsystems (NSUB).**
- Every project should measure its
    - inputs (resources),
    - outputs (deliverables), and
    - results (effectiveness of deliverables).

Reference: *Software Engineering: A Practitioner's Approach, 7/e*

**by Roger S. Pressman**

# Chapter 25

## Process and Project Metrics

*Slide Set to accompany*

*Software Engineering: A Practitioner's Approach, 7/e*

**by Roger S. Pressman**

# References

Software Engineering –
A Practitioner's Approach – 7th ed

By
Roger S Pressman