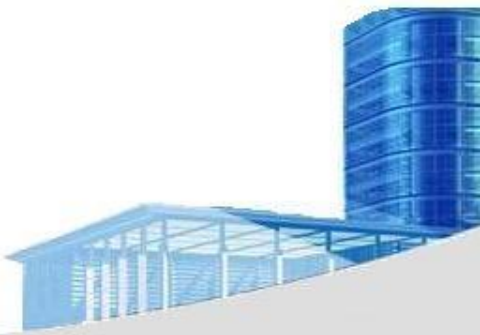
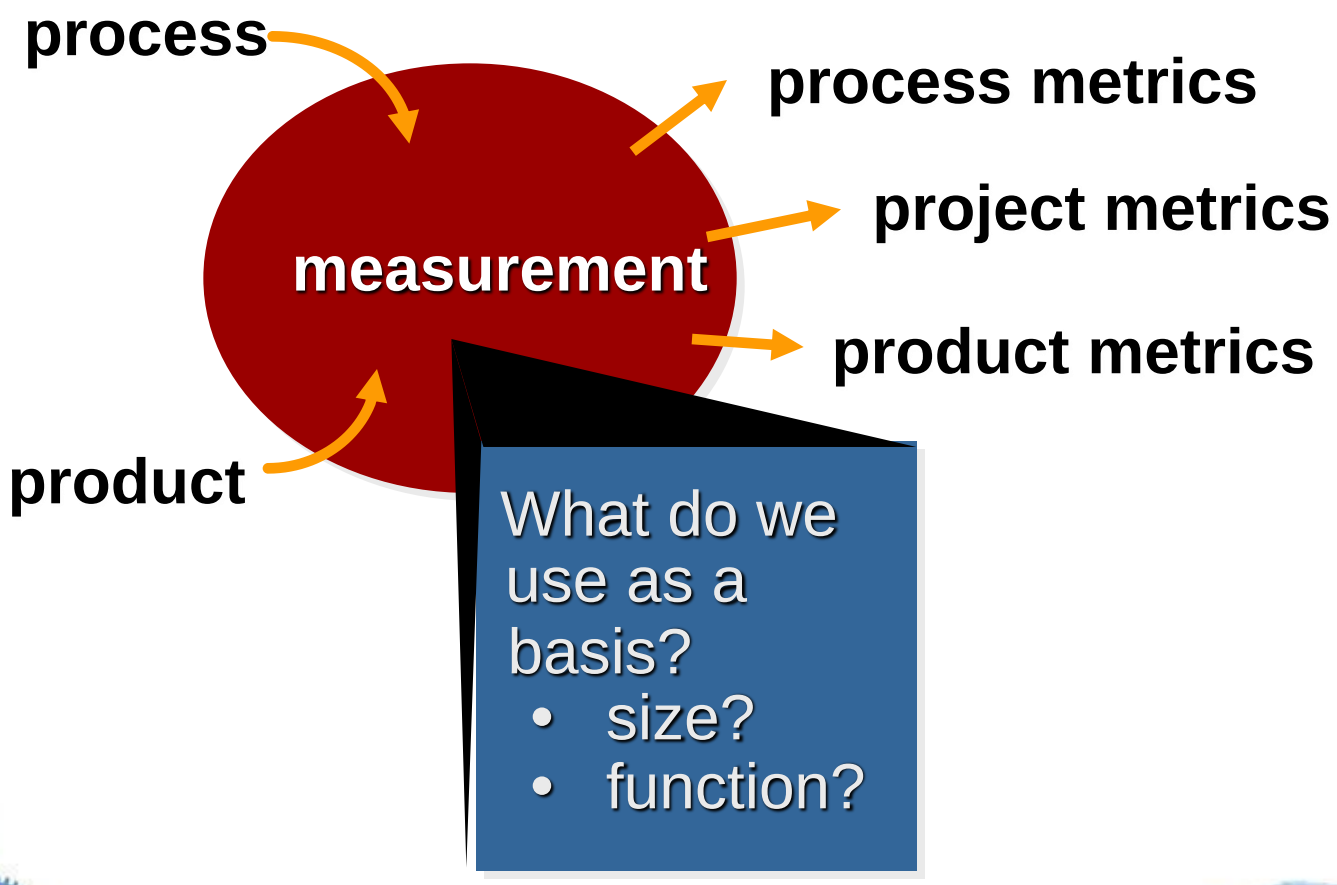
Ch.32 Process and Project Metrics



- **A Good Manager Measures**



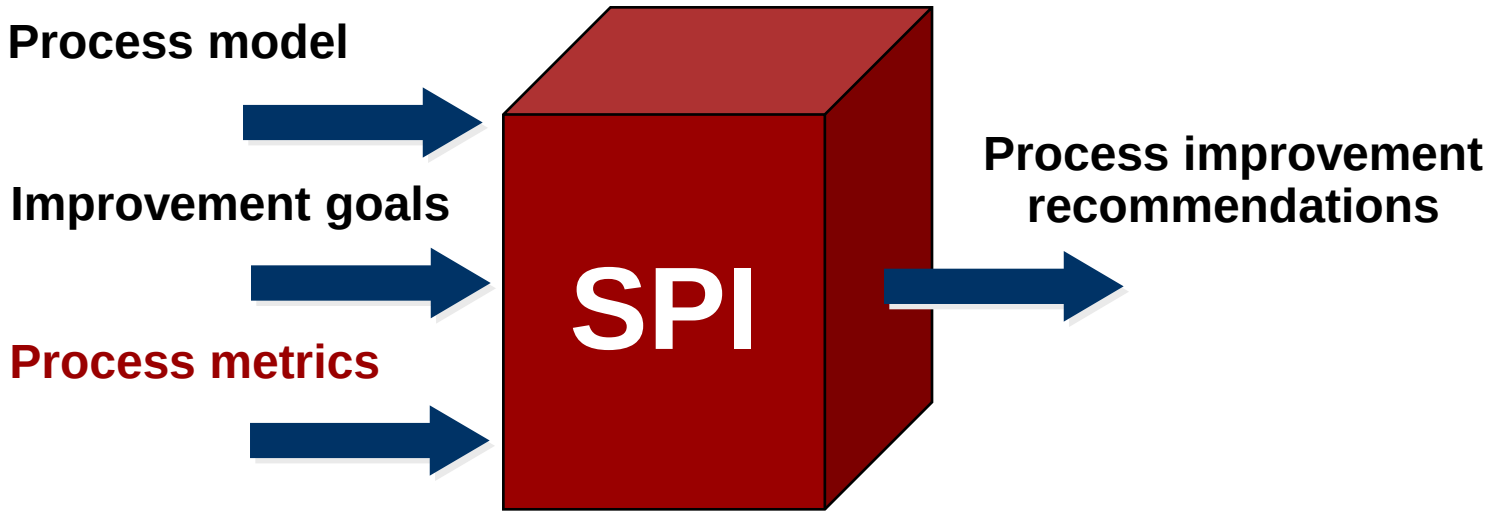
- **Why Do We Measure?**

  - 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.

- **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

- **Process 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	-
JavaSc ript	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.