Chapter 32

- **Process and Project Metrics**

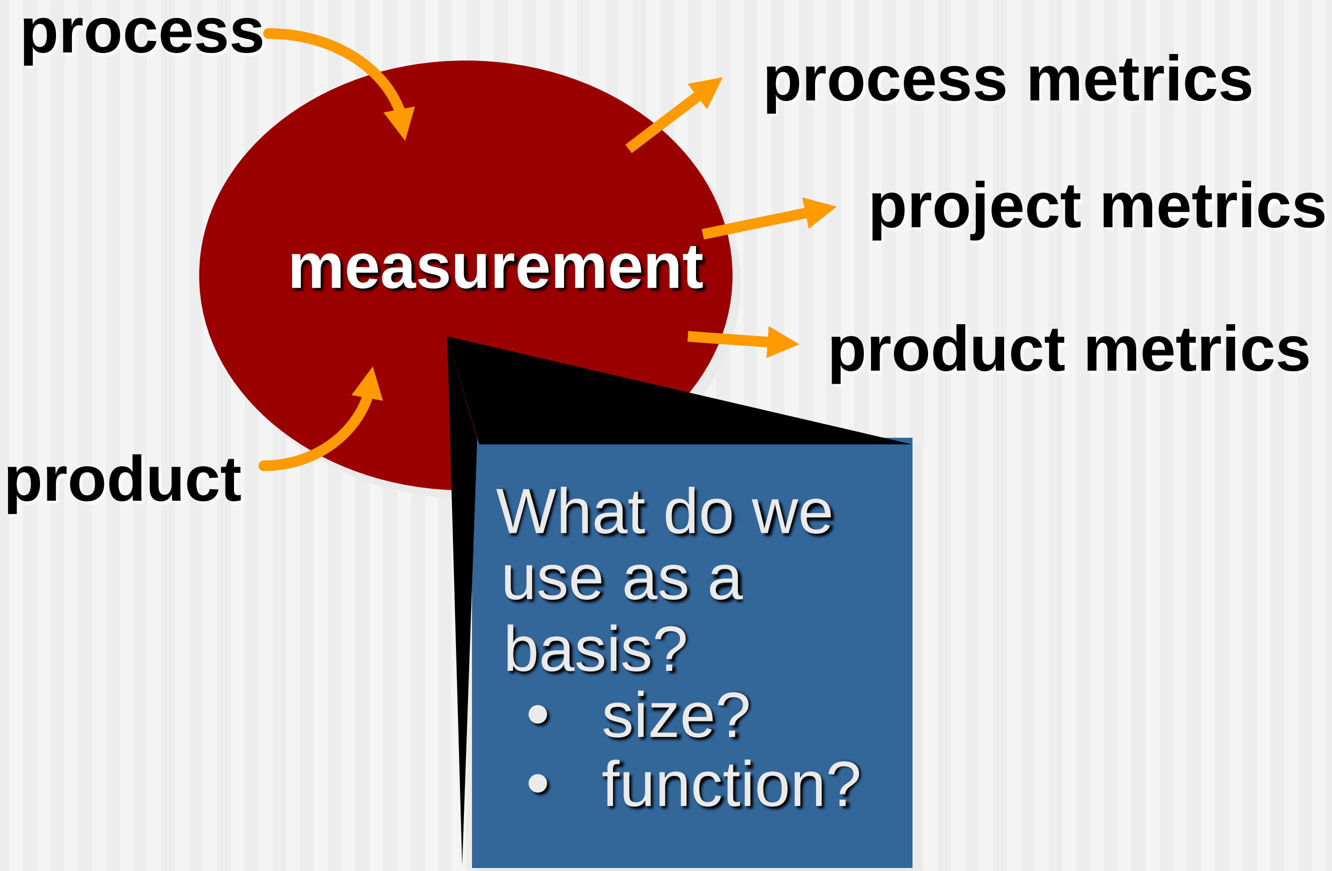
Slide Set to accompany

Software Engineering: A Practitioner's Approach

by Roger S. Pressman

For non-profit educational use only

A Good Manager Measures



What do we use as a basis?

- size?
- function?

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.

Table of Contents

32.1 Metrics in the Process and Project Domains

32.2 Software Measurement

32.3 Metrics for Software Quality

32.4 Integrating Metrics within the Software Process

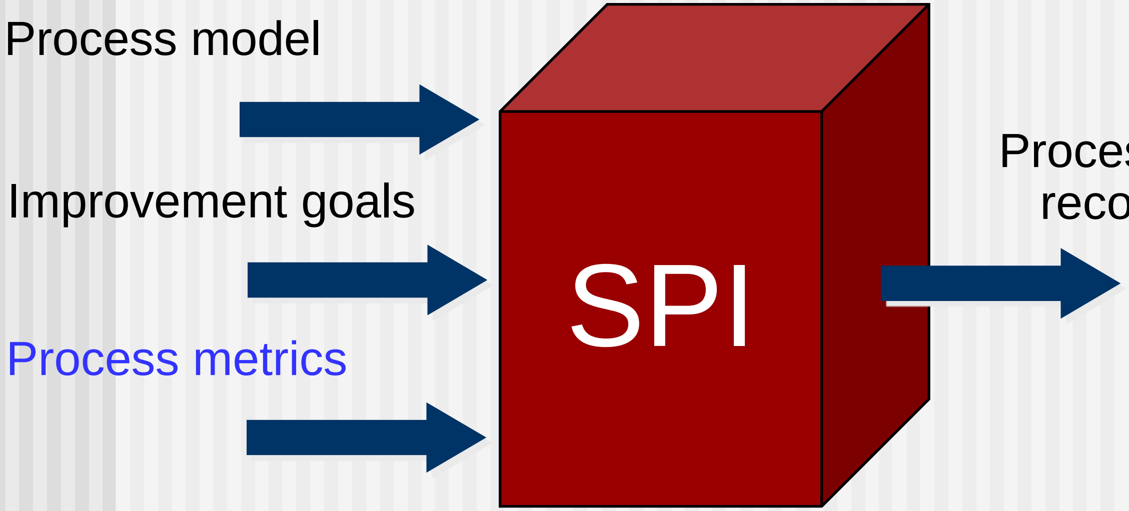
32.5 Metrics for Small Organizations

32.6 Establishing a Software Metrics Program

32.1 Metrics in the Process and Project Domains
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.

32.1.1 Process Metrics and Software Process Improvement


Process model

Improvement goals

Process metrics

SPI

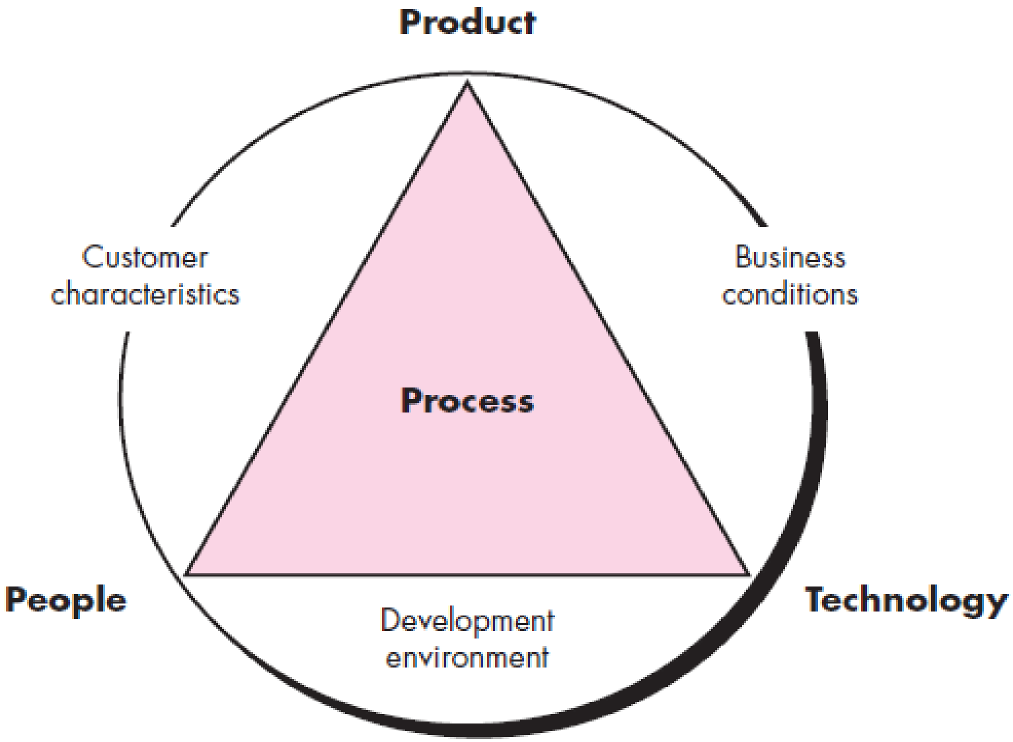
Process improvement recommendations

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 7/e* (McGraw-Hill 2009). Slides copyright 2009 by Roger Pressman.

FIGURE 25.1

Determinants for software quality and organizational effectiveness.

Source: Adapted from [Pau94].



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

32.1.2 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

32.2 Software Measurement

32.2.1 Size-Oriented Metrics

32.2.2 Function-Oriented Metrics

32.2.3 Reconciling LOC and FP Metrics

32.2.4 Object-Oriented Metrics

32.2.5 Use Case-Oriented Metrics

Metrics Guidelines

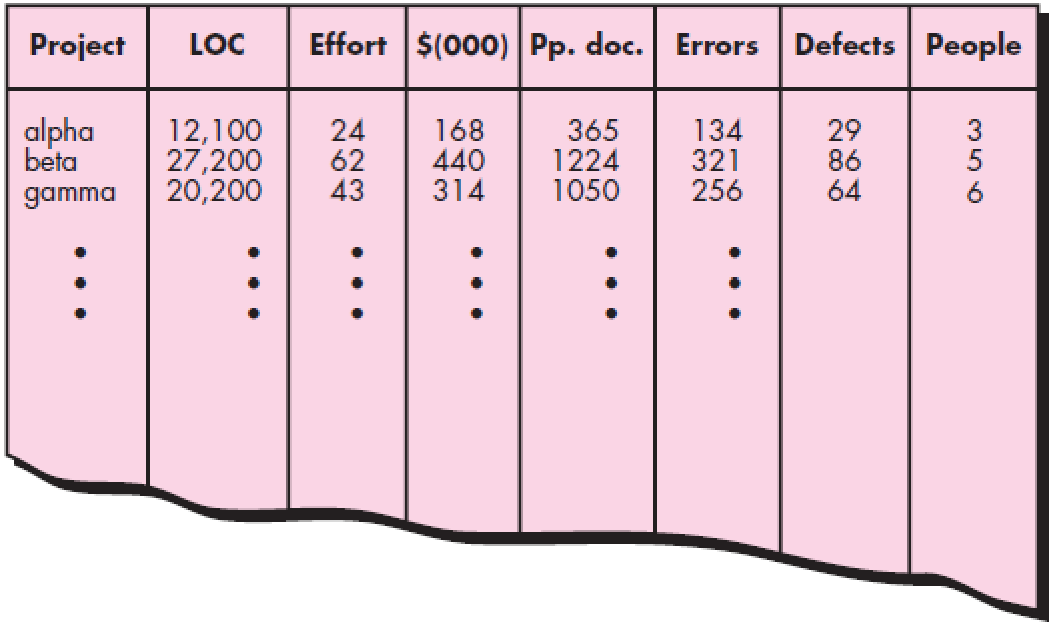
- Use common sense and organizational sensitivity when interpreting metrics data.
- Provide regular feedback to the individuals and teams who have worked to collect measures and metrics.
- Don’t use metrics to appraise individuals.
- Work with practitioners and teams to set clear goals and metrics that 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.

32.2.1 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

FIGURE 25.2

Size-oriented metrics

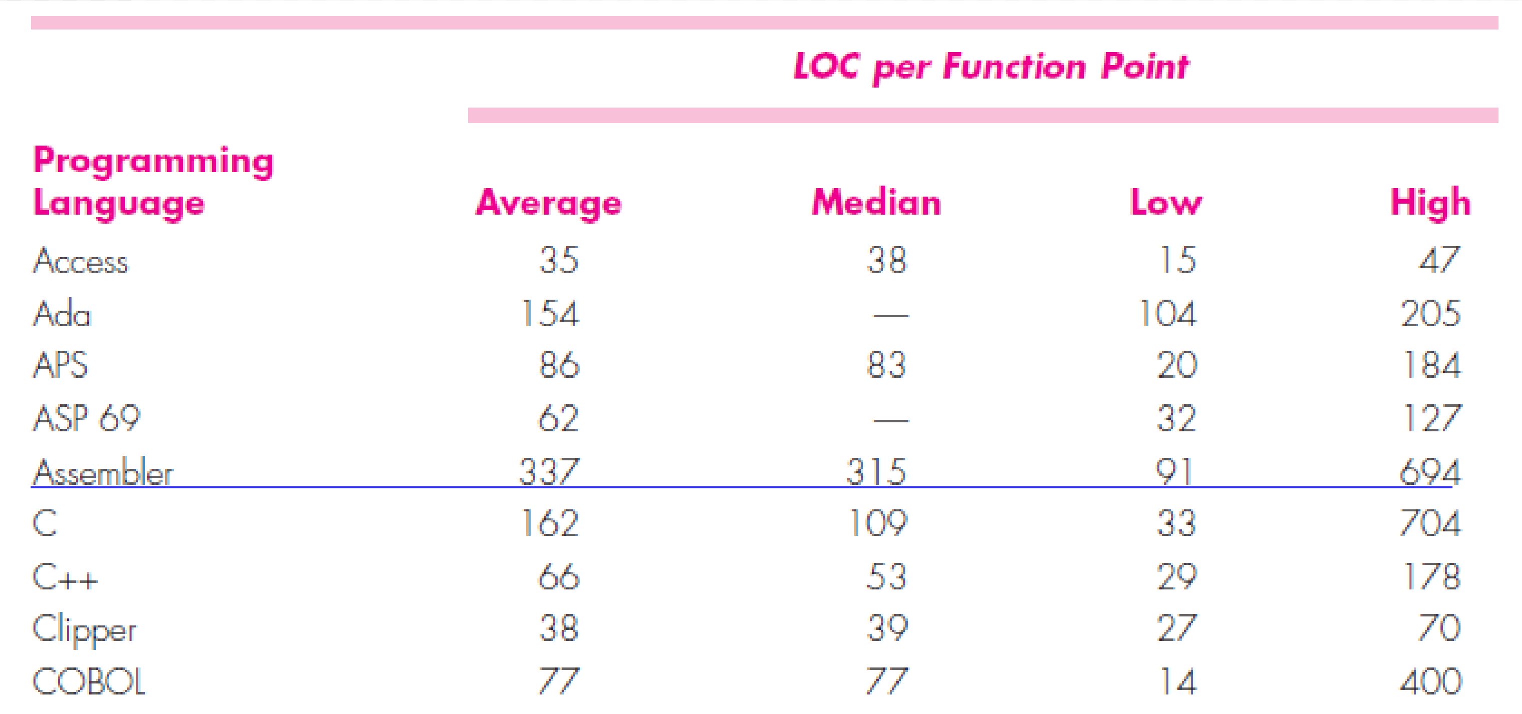


Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
⋮	⋮	⋮	⋮	⋮	⋮		

32.2.2 Function-Oriented Metrics

- Errors per FP (Thousand lines of code)
- Defects per FP
- $ per FP
- Pages of documentation per FP
- FP per person-month

32.2.3 Reconciling LoC and FP Metrics



Programming Language	LOC per Function Point Average	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

FPs can be used to estimate LOC depending on the
AVerage number of LOC (AVC) per FP for a given language
LOC = AVC * number of function points
AVC: 200 ~ 300 for assemble language
2 ~ 40 for a 4GL

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

32.2.4 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)

32.2.5 Use Case-Oriented Metrics

- Researchers have suggested *use-case points* (UCPs) as a mechanism for estimating project effort and other characteristics.
- The UCP is a function of the number of actors and transactions implied by the use-case models and is analogous to the FP in some ways.

32.3 Metrics for Software Quality

32.3.1 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

32.3.2 Defect Removal Efficiency

$$\mathbf{DRE} = \boldsymbol{E\,/(E + D)}$$

where:

E: Number of errors found before delivery

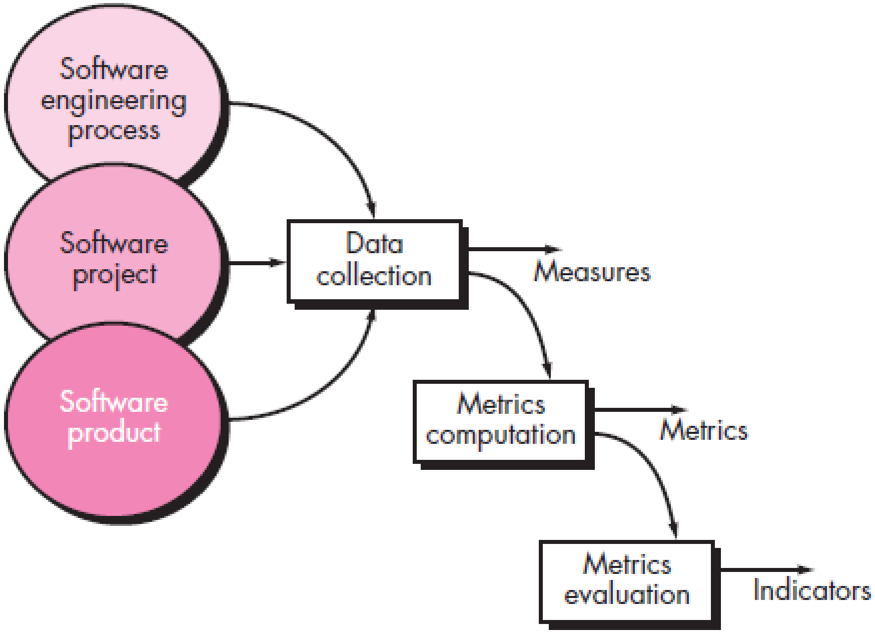
D: Number of defects found after delivery.

32.4 Integrating Metrics within the Software Process

- Arguments for software metrics
  - If you do not measure, there is no real way of determining whether you are improving. And if you are not improving, you are lost.
- Establishing a baseline
  - The metrics baseline consists of data collected from past software development projects
- Metrics Collection, Computation and Evaluation

FIGURE 25.3

Software metrics collection process



32.5 Metrics for Small Organizations

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

32.6 Establishing a Software Metrics Program

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