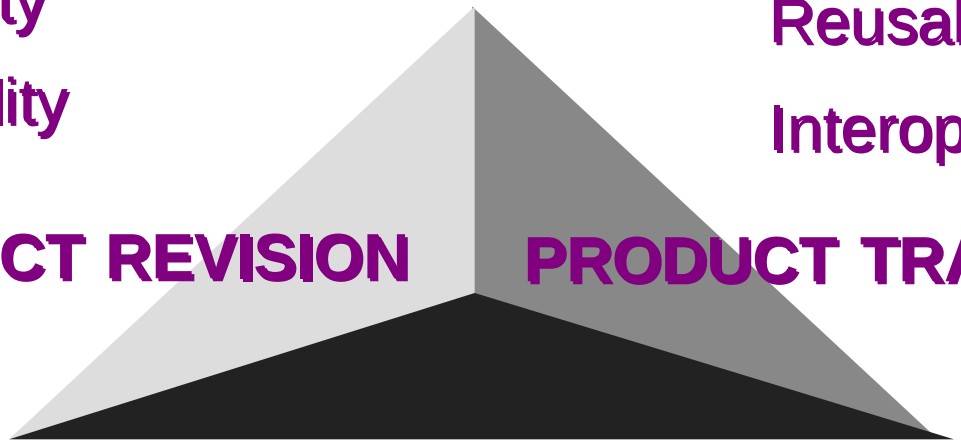
Process and Project Metrics

Adapted from Pressman and schach

McCall's Triangle of Quality

Maintainability
Flexibility
Testability

Portability
Reusability
Interoperability



PRODUCT REVISION

PRODUCT TRANSITION

PRODUCT OPERATION

Correctness
Reliability
Usability
Integrity
Efficiency

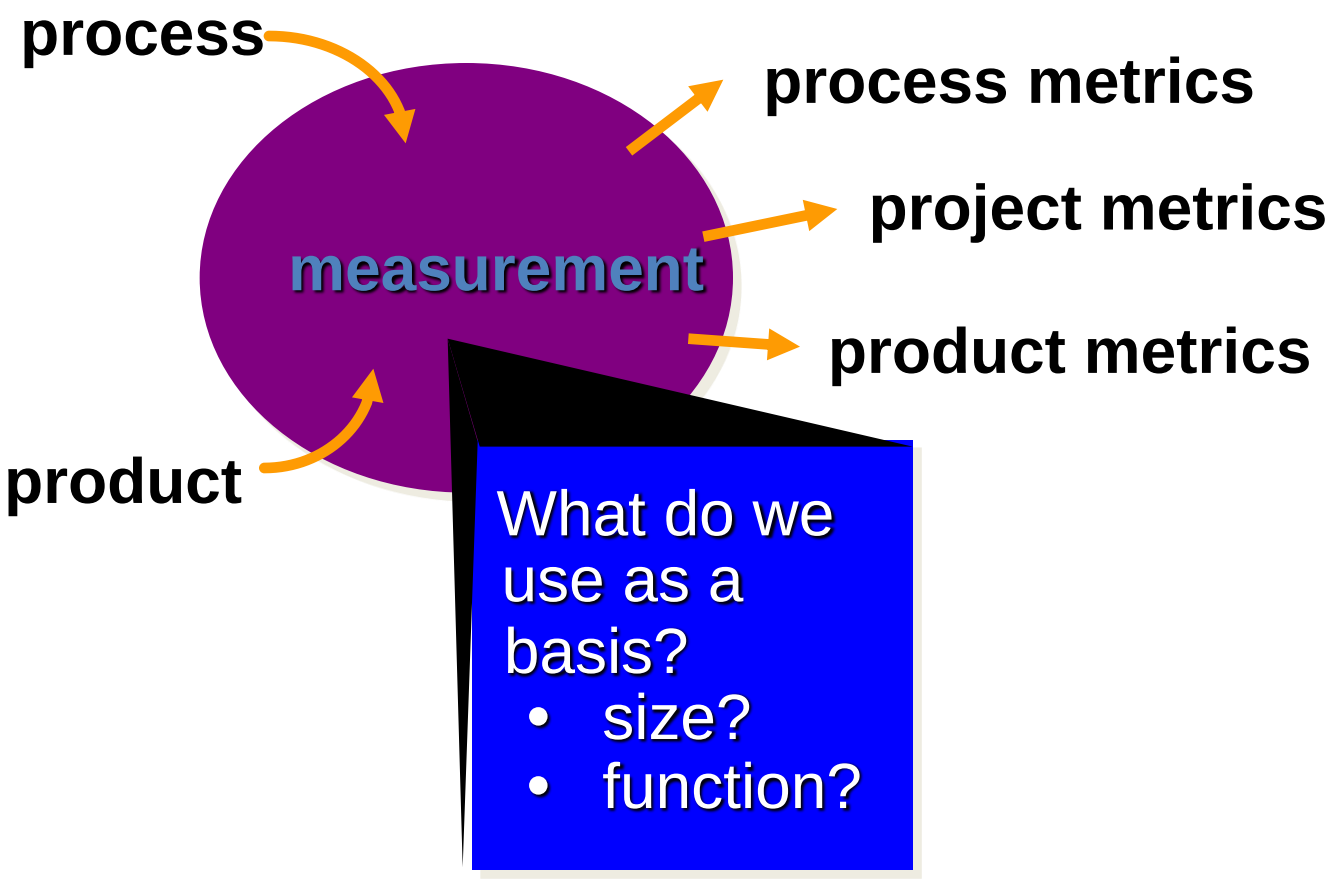
A Comment

McCall's quality factors were proposed in the early 1970s. They are as valid today as they were in that time. It's likely that software built to conform to these factors will exhibit high quality well into the 21st century, even if there are dramatic changes in technology.

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
- The IEEE glossary defines 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 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.

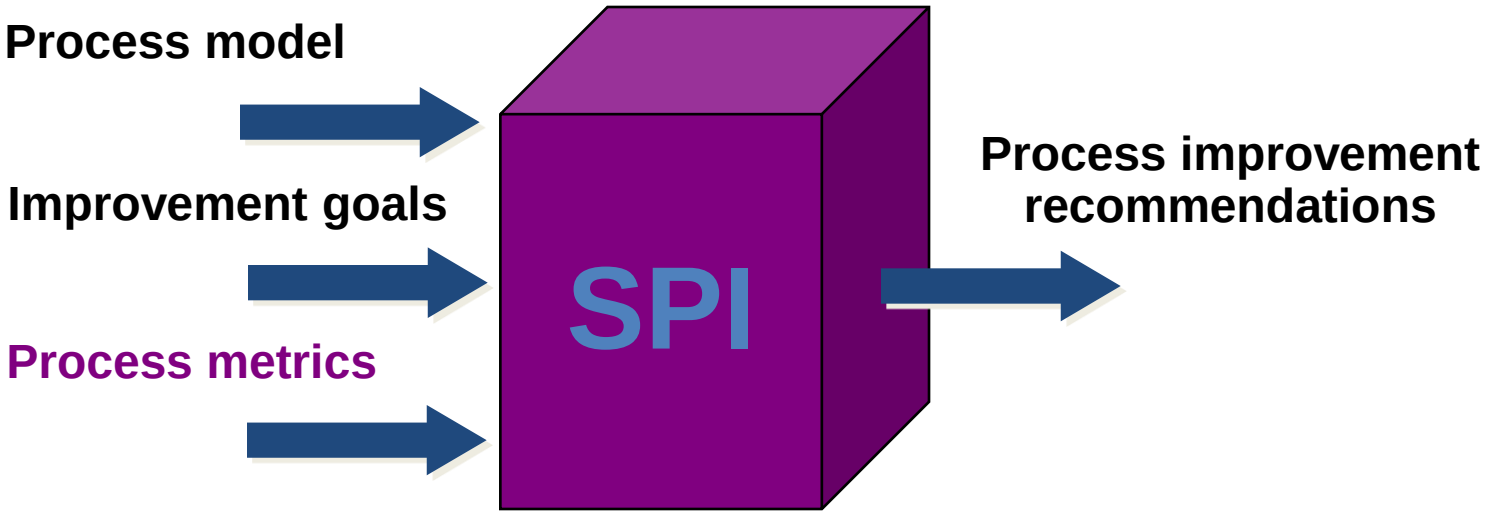
Process Measurement

- 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

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

Lines of Code (LOC)

- Alternate metric
  - Thousand delivered source instructions (KDSI)
- Source code is only a small part of the total software effort
- Different languages lead to different lengths of code
- LOC is not defined for nonprocedural languages (like LISP)

Lines of Code (contd)

- It is not clear how to count lines of code
  - Executable lines of code?
  - Data definitions?
  - Comments?
  - JCL statements?
  - Changed/deleted lines?
- Not everything written is delivered to the client
- A report, screen, or GUI generator can generate thousands of lines of code in minutes

Lines of Code (contd)

- LOC is accurately known only when the product finished
- Estimation based on LOC is therefore doubly dangerous
  - To start the estimation process, LOC in the finished product must be estimated
  - The LOC estimate is then used to estimate the cost of the product — an uncertain input to an uncertain cost estimator

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

Function-Based Metrics

- The *function point metric* (FP), first proposed by Albrecht [ALB79], can be used effectively as a means for measuring the functionality delivered by a system.
- Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity
- Information domain values are defined in the following manner:
  - number of external inputs (EIs)
  - number of external outputs (EOs)
  - number of external inquiries (EQs)
  - number of internal logical files (ILFs)
  - Number of external interface files (EIFs)

Function Points (contd)

- Step 1. Classify each component of the product (*Inp*, *Out*, *Inq*, *Maf*, *Inf*) as simple, average, or complex
  - Assign the appropriate number of function points
  - The sum gives *UFP* (unadjusted function points)

	Level of Complexity		
Component	**Simple**	**Average**	**Complex**
Input item	3	4	6
Output item	4	5	7
Inquiry	3	4	6
Master file	7	10	15
Interface	5	7	10

Function Points (contd)

- Step 2. Compute the technical complexity factor (*TCF*)
  - Assign a value from 0 ("not present") to 5 ("strong influence throughout") to each of 14 factors such as transaction rates, portability

1. Data communication
2. Distributed data processing
3. Performance criteria
4. Heavily utilized hardware
5. High transaction rates
6. Online data entry
7. End-user efficiency
8. Online updating
9. Complex computations
10. Reusability
11. Ease of installation
12. Ease of operation
13. Portability
14. Maintainability

Figure 9.4

Function Points (contd)

- Add the 14 numbers
  - This gives the total degree of influence (*DI*)

$$TCF = 0.65 + 0.01 \times DI$$

- The technical complexity factor (*TCF*) lies between 0.65 and 1.35

Function Points (contd)

- Step 3. The number of function points (*FP*) is then given by

$$FP = UFP \times TCF$$

Analysis of Function Points

- Function points are usually better than KDSI — but there are some problems

- "Errors in excess of 800% counting KDSI, but *only* 200% in counting function points" [Jones, 1987]

Analysis of Function Points

- We obtain nonsensical results from metrics
  - KDSI per person month and
  - Cost per source statement

	Assembler Version	Ada Version
Source code size	70 KDSI	25 KDSI
Development costs	$1,043,000	$590,000
KDSI per person-month	0.335	0.211
Cost per source statement	$14.90	$23.60
Function points per person-month	1.65	2.92
Cost per function point	$3,023	$1,170

Figure 9.5

- Cost per function point is meaningful

Analysis of Function Points

- It is possible to make major changes without changing
  - The number of files, flows, and processes; or
  - The number of inputs, outputs, inquiries, master files, and interfaces

- In theory, it is possible to change every line of code without changing the number of lines of code

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

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}.