Software Metrics

Measurement

- Measurement is fundamental to any engineering discipline
- Software Metrics - Broad range of measurements for computer software
- Software Process - Measurement can be applied to improve it on a continuous basis
- Software Project - Measurement can be applied in estimation, quality control, productivity assessment & project control
- Measurement can be used by software engineers in decision making.

Definitions

- **Measure** - Quantitative indication of the extent, amount, dimension, capacity or size of some attribute of a product or process
- **Measurement** - The act of determining a measure
- **Metric** - A quantitative measure of the degree to which a system, component, or process possesses a given attribute (IEEE Standard Glossary of Software Engineering Terms)

Definitions

- **Indicator** – An indicator is a metric or combination of metrics that provide insight into the software process, a software project or the product itself.

Why Do We Measure?

- To indicate the quality of the product.
- To assess the productivity of the people who produce the product
- To assess the benefits derived from new software engineering methods and tools
- To form a baseline for estimation
- To help justify requests for new tools or additional training

Types of Metrics

1. Process Metrics

3. Product Metrics

5. Project Metrics

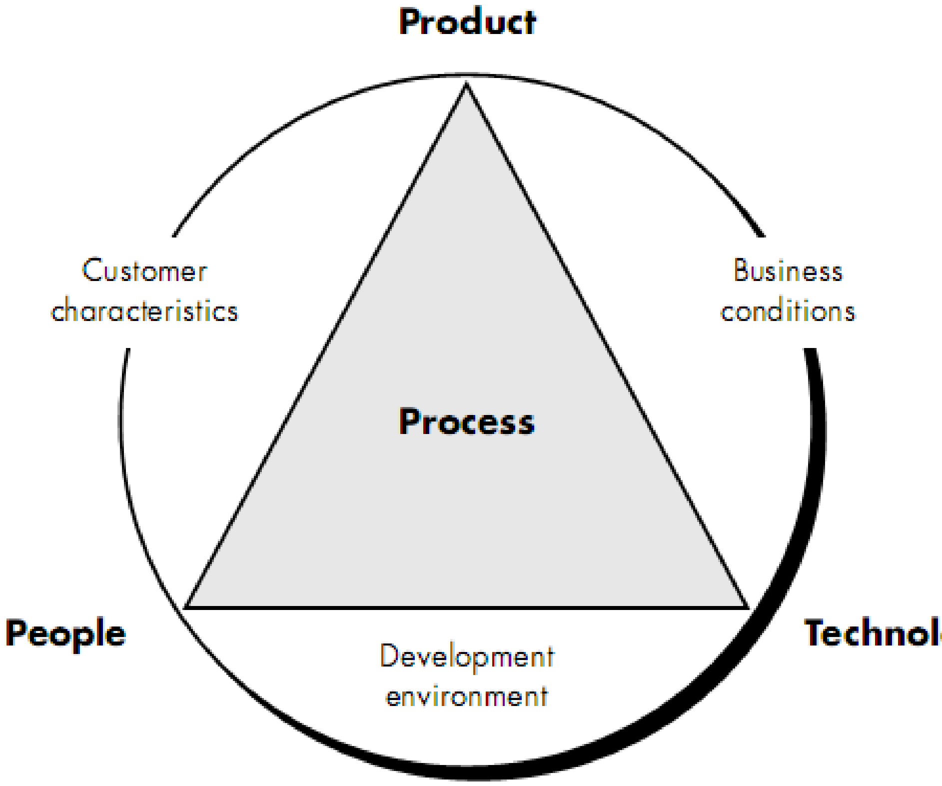
Process Metrics

- Process metrics are measures of the software development process, such as
  - Overall development time
  - Type of methodology used
- Process metrics are collected across all projects and over long periods of time.
- Their intent is to provide indicators that lead to long-term software process improvement.

Process Metrics & Software Process Improvement

- To improve any process, the rational way is:
  - Measure Specific attributes of the process
  - Derive meaningful metrics from these attributes.
  - Use these metrics to provide indicators.
  - The indicators lead to a strategy for improvement.

Factors Affecting Software Quality



How to Measure Effectiveness of a Software Process

- We measure the effectiveness of a software process indirectly
- We derive a set of metrics based on the outcomes that can be derived from the process.
- Outcomes include
  - 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 etc.
  - Conformance to schedule

Project Metrics

- Project Metrics are the measures of Software Project and are used to monitor and control the project. They enable a software project manager to:

  - Minimize the **development time** by making the adjustments necessary to avoid delays and potential problems and risks.

  - Assess **product quality** on an ongoing basis & modify the technical approach to improve quality.

Project Metrics

- Used in estimation techniques & other technical work.
- Metrics collected from past projects are used as a basis from which effort and time estimates are made for current software project.
- As a project proceeds, actual values of human effort & calendar time expended are compared to the original estimates.
- This data is used by the project manager to monitor & control the project.

Product metrics

- Product metrics are measures of the software product at any stage of its development, from requirements to installed system. Product metrics may measure:
  - the complexity of the software design
  - the size of the final program
  - the number of pages of documentation produced

Types of Software Measurements

- **Direct measures**
  - Easy to collect
  - E.g. Cost, Effort, Lines of codes (LOC), Execution Speed, Memory size, Defects etc.
- **Indirect measures**
  - More difficult to assess & can be measured indirectly only.
  - Quality, Functionality, Complexity, Reliability, Efficiency, Maintainability etc.

An example

- 2 different project teams are working to record errors in a software process
- Team A – Finds 342 errors during software process before release
- Team B- Finds 184 errors

- Which team do you think is more effective in finding errors?

Normalization of Metrics

- To answer this we need to know the size & complexity of the projects.
- But if we normalize the measures, it is possible to compare the two
- For normalization we have 2 ways-
  - Size-Oriented Metrics
  - Function Oriented Metrics

Size-Oriented Metrics

- Based on the “size” of the software produced

Size-Oriented Metrics

Project	Effort (person-month)	Cost ($)	LOC	kLOC	Doc. (pgs)	Errors	People
A	24	168,000	12100	12.1	365	29	3
B	62	440,000	27200	27.2	1224	86	5

From the above data, simple size oriented metrics can be developed for each Project

- Errors per KLOC
- $ per KLOC
- Pages of documentation per KLOC
- Errors per person-month
- LOC per person-month
- **Advantages of Size Oriented Metrics**
  - LOC can be easily counted
  - Many software estimation models use LOC or KLOC as input.
- **Disadvantages of Size Oriented Metrics**
  - LOC measures are language dependent, programmer dependent
  - Their use in estimation requires a lot of detail which can be difficult to achieve.
- Useful for projects with similar environment

Function-Oriented Metrics

- Based on “functionality” delivered by the software
- Functionality is measured indirectly using a measure called *function point*.
- Function points (FP) - derived using an empirical relationship based on countable measures of software & assessments of software complexity

Steps In Calculating FP

1. Count the measurement parameters.
2. Assess the complexity of the values.
3. Calculate the raw FP (see next table).
4. Rate the complexity factors to produce the complexity adjustment value (CAV)
5. Calculate the adjusted FP as follows:

$$\mathbf{FP = raw\ FP \times [0.65 + 0.01 \times CAV]}$$

Function Point Metrics

Parameter	Count		Simple	Average	Complex		
Inputs		x	3	4	6	=	
Outputs		x	4	5	7	=	
Inquiries		x	3	4	6	=	
Files		x	7	10	15	=	
Interfaces		x	5	7	10	=	

Count-total (raw FP)

Software information domain values

- Number of user inputs
- Number of user outputs
- Number of user inquiries
- Number of files
- Number of external interfaces

Rate Complexity Factors

For each **complexity adjustment factor**, give a rating on a scale of 0 to 5

- 0 - No influence
- 1 - Incidental
- 2 - Moderate
- 3 - Average
- 4 - Significant
- 5 - Essential

Complexity Adjustment Factors

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?

Complexity Adjustment Factors(Continue…)

1. Are the master files updated on-line?
2. Are the inputs, outputs, files, or inquiries complex?
3. Is the internal processing complex?
4. Is the code designed to be reusable?
5. Are conversion and installation included in the design?
6. Is the system designed for multiple installations in different organizations?
7. Is the application designed to facilitate change and ease of use by the user?

Complexity Adjustment Value

- The rating for all the factors, F_1 to F_{14}, are summed to produce the complexity adjustment value (CAV)
- CAV is then used in the calculation of the function point (FP) of the software

Example of Function-Oriented Metrics

- Errors per FP
- Defects per FP
- $ per FP
- Pages of documentation per FP
- FP per person month

FP Characteristics

- Advantages: language independent, based on data known early in project, good for estimation
- Disadvantages: calculation complexity, subjective assessments, FP has no physical meaning (just a number)

Qualities of a good metric

- simple, precisely definable—so that it is
- clear how the metric can be evaluated;
- objective, to the greatest extent possible;
- easily obtainable (i.e., at reasonable cost);
- valid—the metric should measure what it
- is intended to measure; and
- robust—relatively insensitive to (intuitive-
- ly) insignificant changes in the process or
- product.