BAIT3153 Software and Project Management (SPM)

Software Metrics & Measurements

Chapter 4

Table of Contents

Metrics - *numbers that give you information about a particular process/activity/product/etc*

METRICS | English meaning - Cambridge Dictionary
https://dictionary.cambridge.org

4.1 Introduction

- Software measurement is concerned with deriving a numeric value for quantitative evaluation of an attribute of a software product or process.
- Applied in software process with the intent of improving it on a continuous basis.
- This allows for objective comparisons between products, techniques and processes.

Purpose of measurements

4.1 Introduction

- To assign a value to system quality attributes
  - By measuring the characteristics of system components, such as their cyclomatic complexity, and then aggregating these measurements, you can **assess** **system quality** attributes, such as maintainability.
- To identify the system components whose quality is sub-standard
  - Measurements can identify individual components with characteristics that deviate from the norm. For example, you can measure components to discover those with the highest **complexity**. These are most likely to contain bugs because the complexity makes them harder to understand.

4.2 Software Metrics

- **Examples of Software Metrics**
- Use of Software Metrics
- Indicators
- Types of Metrics

4.2 Software Metrics

- A quantitative measure of the degree to which a system, component or process possesses a given attribute.
- **Examples of software metrics** (*below are normalized metrics*)**:**
  - **Errors/KLOC**
  - **Defects/KLOC**
  - **Cost/LOC**
  - **Pages of document/KLOC**
  - **Errors/person-month**
  - **LOC/person-month**

	Non normalized			
	Project A	Project B	Project A	Project B
Total Line of Code (LOC)	306,000	256,000	-	-
Errors	288	233		
Defects	30	20		
Pages of Documentation	205	180		

P(A) has 50% more defects than P(B)

4.2 Software Metrics

- A quantitative measure of the degree to which a system, component or process possesses a given attribute.
- **<u>Examples of software metrics</u> (*below are normalized metrics*):**
  - **Errors/KLOC**
  - **Defects/KLOC**
  - **Cost/LOC**
  - **Pages of document/KLOC**
  - **Errors/person-month**
  - **LOC/person-month**

	Non normalized		**Normalized with KLOC**	
	Project A	**Project B**	Project A	Project B
Total Line of Code (LOC)	**306,000**	**256,000**	-	-
Errors	**288**	**233**	233 / 306 = 0.9412	233 * / 256 = 0.9102
Defects	**30**	**20**	30 / 3[illegible]6 = 0.09[illegible]	20 / 256 = 0.0781
Pages of Documenta[illegible]	**205**	**180**	2[illegible]6 =	180/233 =

P(A) has 50% more defects than P(B)

P(A) has 25% more defects than P(B)

Use of Software Metrics

4.2 Software Metrics

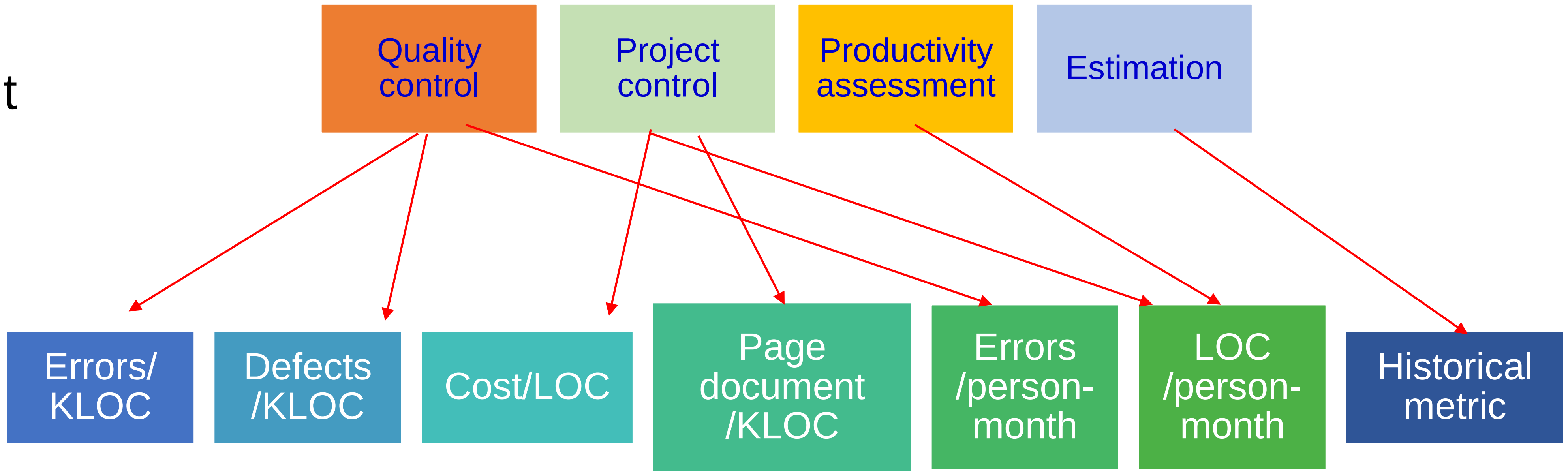
- **Quality control**
  - Measures of the fitness of use of the work products that are produced.
- **Project control**
- **Productivity assessment**
  - Measures of software development output as a function of effort and time applied.
- **Estimation (using historical metrics)**
  - What was software development productivity on past projects?
  - What was the quality of the software that was produced?
  - How can past productivity and quality data be extrapolated to the present?
  - How can it help us plan and estimate more accurately?

Exercise:

Match the following examples of metrics to the purpose of measurement:

Purpose of measurement

Metrics



Thousand Line of Code : KLOC

Indicators

4.2 Software Metrics

	Non normalized		Normalized with KLOC	
	Project A	Project B	Project A	Project B
Total Line of Code (LOC)	306,000	256,000	-	-
Errors	288	233	233 / 306 = 0.9412	233 * / 256 = 0.9102
Defects	30	20	30 / 306 = 0.098	20 / 256 = 0.0781
Page of Documentation	205	180		

The normalized metrics, give you an indicator that Project A quality is lower than Project B !

- A metric or combination of metrics that provides insight into the software process, project or the product itself so that improvement or adjustment can be made to the process or project.
- **E.g.** 800 errors/KLOC is an indicator depicting an insight that the software is of poor quality.

Indicators

4.2 Software Metrics

	Non normalized		Normalized with KLOC	
	Project A	Project B	Project A	Project B
Total Line of Code (LOC)	306,000	256,000	-	-
Errors	288	233	233 / 306 = 0.9412	233 * / 256 = 0.9102
Defects	30	20	30 / 306 = 0.0980	20 / 256 = 0.0781
Page of Documentation	205	180	205/306 = 0.6699	180/256 = 0.7031

The normalized metrics highlighted here, give you an indicator that Project B produces more pages of project documents than Project A! *(more pages of project documents give you more info about the project)*

- A metric or combination of metrics that provides insight into the software process, project or the product itself so that improvement or adjustment can be made to the process or project.
- E.g. 800 errors/KLOC is an indicator depicting an insight that the software is of poor quality.

Types of Metrics

4.2 Software Metrics

Category of Software metrics:

a. **Product metrics**
    - Dynamic metrics
    - Static metrics

b. **Process metrics**
    - Time, resources, events
    - Public
    - Private

c. **Project metrics**

a. Product metrics

- **Concerned with the quality of the software itself**
- 2 Classes of product metric
  - Dynamic metrics
    - Collected by measurements made of a program in execution
    - Help assess efficiency and reliability
  - Static metrics
    - Collected by measurements made of the system representations such as design, program or documentation
    - Help assess complexity, understandability and maintainability.

Dynamic Metrics

- Closely related to software quality attributes
- **Measurements of a program/system in execution**
- **Examples** of measurement:
  - time required **to start up the system,**
  - execution time required **for a particular function** (performance/efficiency attribute)
  - **the number of** system failures (reliability attribute).
- **Can be collected during** system testing **or after the** system has gone into use

a. **Product metrics**

4.2 Software Metrics

- **Concerned with the quality of the software itself**
- 2 Classes of product metric
  - Dynamic metrics
    - Collected by measurements made of a program in execution
    - Help assess efficiency and reliability
  - Static metrics
    - Collected by measurements made of the system representations such as design, program or documentation
    - Help assess complexity, understandability and maintainability.

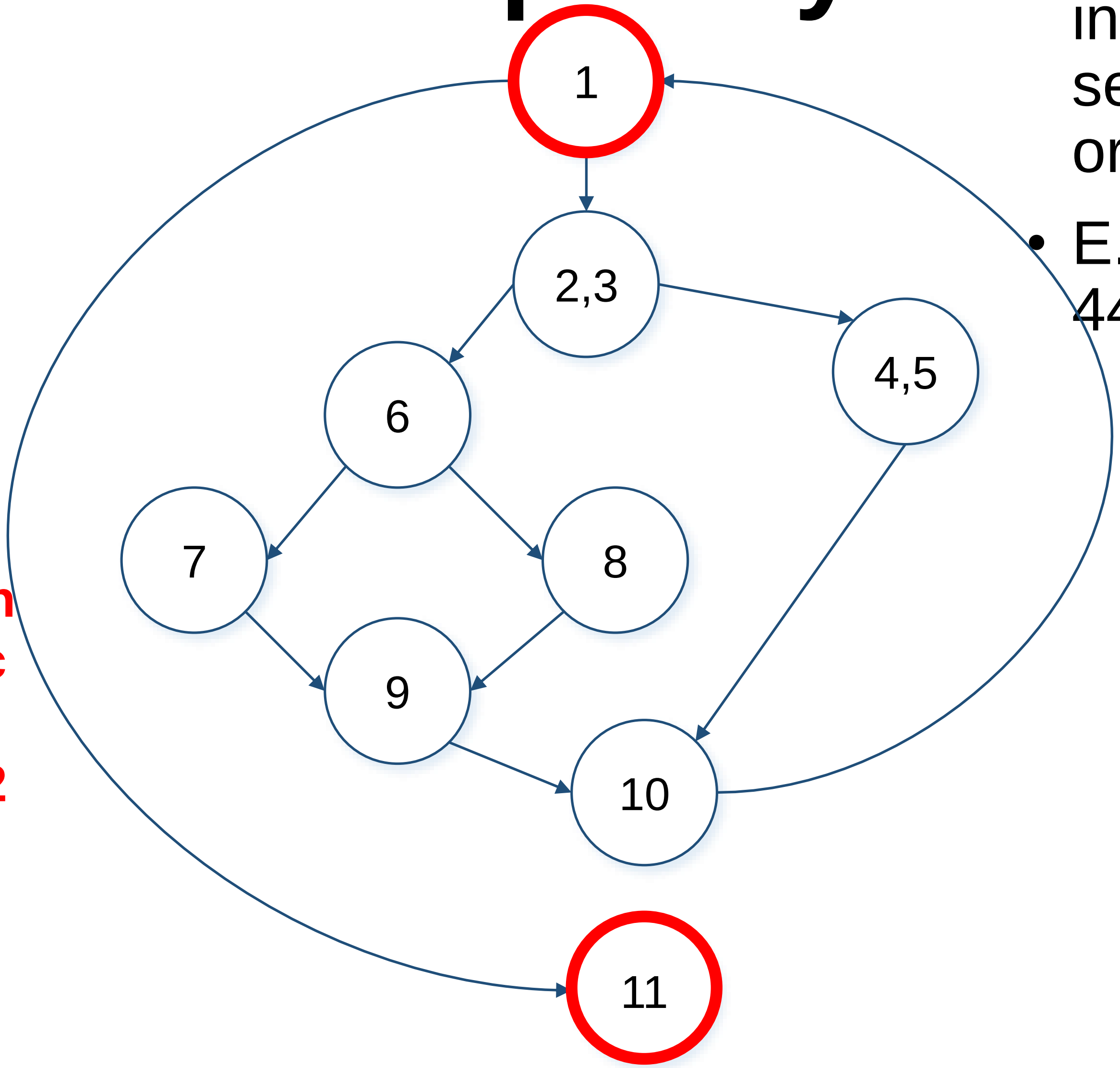
Static Metrics

- Have an indirect relationship with quality attributes. You need to try and derive a relationship between these metrics and properties such as complexity, understandability and maintainability.
- **Measurements made of the system representations such as design, program or documentation**
- **Examples:**
  - Line of code (LOC) – **Measure size of a program. more lines, more complex, more errors**
  - Depth of conditional nesting – **Measure depth of nested if-statement. Deeply nested if-statement are hard to understand and potentially error-prone**
  - Length of identifiers – longer, more meaningful
  - Cyclomatic complexity – **Measure control complexity of a program**. *(see next slide)*

Cyclomatic Complexity

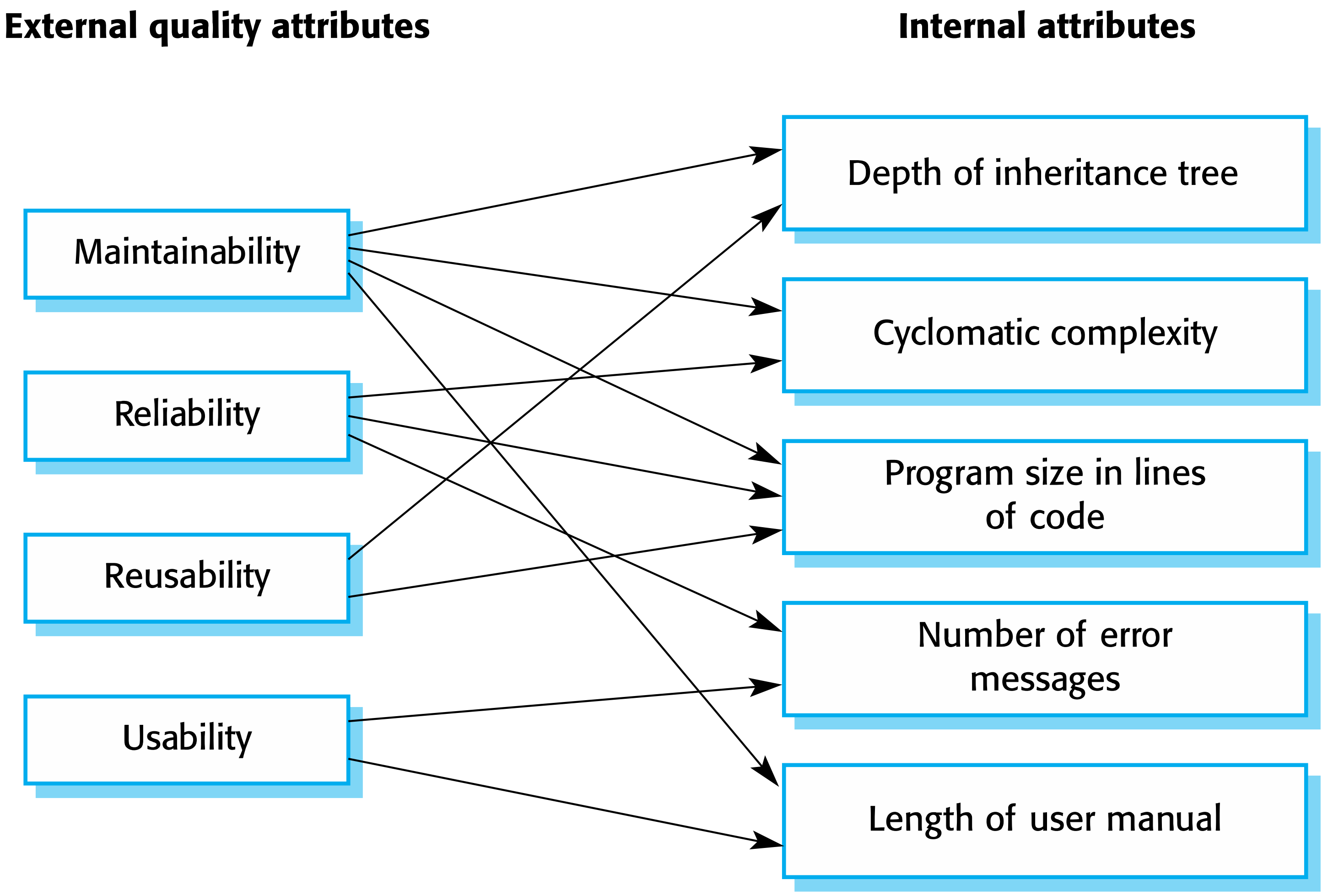
If the source code has only ONE path then cyclomatic complexity = 1

If the source code contains one IF condition then cyclomatic complexity = 2 because there 2 paths



- Independent path: any path through the program that introduces at least one new set of processing statements or a new condition.
- E.g. in **Flowchart** A (ref 3 pg 446), independent paths are:
  - **Path 1: 1-11**
  - **Path 2: 1-2-3-4-5-10-1-11**
  - **Path 3: 1-2-3-6-8-9-10-1-11**
  - **Path 4: 1-2-3-6-7-9-10-1-11**

Relationships between internal and external software quality attributes



Types of Metrics

4.2 Software Metrics

Category of Software metrics:

a. Product metrics
   - Dynamic metrics
   - Static metrics

b. **Process metrics**
   - Time, resources, events
   - Public
   - Private

c. Project metrics

b. Process Metrics

4.2 Software Metrics

- **Gives an indication of the <u>system development processes</u>**
- **Examples:**

• Measures of **errors uncovered** <u>before</u> the release of the software • **Defects <u>delivered</u>** to and **<u>reported</u>** by users • **Work products/deliverables <u>delivered</u>** (productivity) • Schedule conformance • Human effort expended • Calendar time expended	• Time (hr or day)/SE task - can also be project metrics) • Elapsed Time - time a request is received until evaluation is complete • Effort (person-hr) to perform the evaluation • Time elapsed from completion of evaluation to assignment of change order to personnel • <u>Effort</u> required to make the <u>change</u> • <u>Time</u> required to make the <u>change</u> • Errors uncovered during work to make change

Note: Some of the process metrics are also metrics for project and product as well

b. Process Metrics

Types of process metric

- *The time **taken for a particular process to be completed***
  - This can be the total time devoted to the process, calendar time, the time spent on the process by particular engineers, and so on.
- *The resources **required for a particular process***
  - Resources might include total effort in person-days, travel costs or computer resources.
- *The number of occurrences **of a particular event***
  - Examples of events that might be monitored include:
  - **the number of defects discovered during code inspection,**
  - **the number of requirements changes requested,**
  - **the number of bug reported after delivery**
  - **the average number of lines of code modified in response to a requirements change.**

b. Process Metrics

Public vs Private Process Metrics

- Public metrics
  - Integrated information that was originally private to individuals and teams
  - Examples:
    - Project level defect rates
    - Effort
    - Calendar times
    - Related data to uncover indicators to improve organizational process performance

- Private metrics
  - Process metrics that should be private to the individual software engineer and serve as an indicator for the individual only
  - Examples:
    - Defect rates (by individual)
    - Defect rates (by module)
    - Errors found during development

b. Process Metrics

Personal Software Process (PSP)

- An approach that uses private process metrics designed to help software engineers improve their performance
- Provides disciplined methods to help software engineers
  - Improve their estimating and planning skills
  - Make commitments they can keep
  - Manage the quality of their projects
  - Reduce the number of defects in their work
- Each level has detailed scripts, checklists and templates to guide the engineer through required steps and helps them improve their own personal software process.

b. Process Metrics

PSP Core Measures

- Size – the size measure for a product part, e.g. LOC
- Effort – the time required to complete a task (usually in minutes)
- Quality – the number of defects in the product
- Schedule – a measure of project progression, tracked against planned and actual completion dates

Note:

- software developers use many other measures derived from these basic measures, e.g.: estimation accuracy, productivity, PV, EV, etc.
- Logging time, defect and size data is an essential part of planning and tracking PSP projects as historical data is used to improve estimating accuracy.

Types of Metrics

4.2 Software Metrics

Category of Software metrics:

a. Product metrics
   - Dynamic metrics
   - Static metrics

b. Process metrics
   - Time, resources, events
   - Public
   - Private

c. **Project metrics**

c. Project Metrics

4.2 Software Metrics

- **Used by Project Manager and team to**
  - **Monitor progress during software development,**
  - Adapt project workflow and technical activities, and
  - **Control product quality**

- Compare actual metrics with planned metrics to:
  - Make necessary adjustments to avoid delays and mitigate potential problems and risks
  - Assess product quality on an on-going basis and modify the technical approach to improve quality

Collection of Project Metrics

- During estimation:
  - Metrics collected from past projects are used as a basis from which effort and time estimates are made for current work
- **During project execution:**
  - Compare scheduled dates & actual milestone dates to monitor progress
  - **Make necessary adjustments to avoid delays and mitigate potential problems and risks**
- **During technical work:**
  - Count errors uncovered per review hour
  - Distribution of **effort per SE task**
  - Pages of documentation per SE task
  - Function points per SE task
  - Delivered source lines per SE task

Types of Metrics

4.2 Software Metrics

Category of Software metrics:

a. Product metrics

- Dynamic metrics
- Static metrics

b. Process metrics

- Time, resources, events
- Public
- Private

c. Project metrics

***Note**: Some of the process metrics are also metrics for project and product as well*

4.3 Software Measurement

- Direct and Indirect Measures
- **Size-Oriented Metrics**
- **Function-Oriented Metrics**

Direct and Indirect Measures

4.3 Software Measurement

Direct Measure

- more quantitative, easier to collect.
- E.g.:
  - Process & Project
    - Cost & effort applied
  - Product
    - LOC
    - Execution speed
    - Memory size
    - Defects over time

Indirect Measure

- measurement towards non-functional requirements but can still be expressed in quantitative form, quite difficult to measure.
- E.g.:
  - Functionality
  - Quality
  - Complexity
  - Efficiency
  - Reliability
  - Maintainability

Size-Oriented Metrics

- Project A has 20 errors while Project B has 50 errors. Can we conclude that Project A is of better quality?
- Normalization of metrics between projects are required to reduce/get a value that is comparable in a fair manner between different projects.
- Two methods to obtain normalized metrics to compare different projects:
  - Size-oriented metrics
  - Function-oriented metrics

Size-Oriented Metrics

4.3 Software Measurement

Size-oriented metrics are derived by normalizing (dividing) with a related measure, e.g.:

- Quality measures over product size
- Productivity measures over the effort

Quality

Example: $\frac{\text{Errors}}{\text{KLOC}}$

Productivity

Example: $\frac{\text{LOC}}{\text{Person-month}}$

Size-oriented Metrics - **Unnormalized**

Project	LOC	Effort (MTH)	$(000)	pp. doc.	Errors	Defects	People (SE)
Alpha	12,100	24	168	365	134	29	3
Beta	27,200	62	440	1,224	321	86	5
Gamma	20,200	43	314	1,050	256	64	6
…	…	…	…	…	…	…	…

Size-oriented Metrics - **Normalized**

Project	LOC/SE	LOC/MTH	$/LOC		Errors/ KLOC	Defects/ KLOC	People (SE)
Alpha	**12,100/3 = 4,033**	**12,100/24 = 504**	**168,000/ 12100 = 13.88**		**1,340/121 = 11.07**		
Beta	**27,200/5 = 5,440**	**27,200/62 = 438**	**440,000/ 27200 = 16.18**		**3,210/272 = 11.80**		

12100/1000 = 12.1
134/12.1 = 11.07

27200/1000=27.2
321/27.2 = 11.08

Size-oriented Metrics - **Unnormalized**

Project	LOC	Effort (MTH)	$(000)	pp. doc.	Errors	Defects	People (SE)
Alpha	12,100	24	168	365	134	29	3
Beta	27,200	62	440	1,224	321	86	5
Gamma	20,200	43	314	1,050	256	64	6
…	…	…	…	…	…	…	…

Size-oriented Metrics - **Normalized**

Project	LOC/SE	LOC/MTH	$/LOC		Errors/KLOC	Defects/KLOC	People (SE)
Alpha	**12,100/3 = 4,033**	**12,100/24 = 504**	**168,000/12100 = 13.88**		**1,340/121 = 11.07**		
Beta	**27,200/5 = 5,440**	**27,200/62 = 438**	**440,000/27200 = 16.18**		**3,210/272 = 11.80**		

Small differences in quality

Size-oriented Metrics - **Unnormalized**

Project	LOC	Effort (MTH)	$(000)	pp. doc.	Errors	Defects	People (SE)
Alpha	12,100	24	168	365	134	29	3
Beta	27,200	62	440	1,224	321	86	5
Gamma	20,200	43	314	1,050	256	64	6
…	…	…	…	…	…	…	…

Size-oriented Metrics - **Normalized**

Project	LOC/SE	LOC/MTH	$/LOC			…cts/ …C	People (SE)
Alpha	**12,100/3 = 4,033**	**12,100/24 = 504**	**168,000/ 12100 = 13.88**		**1 = 11.07**		
Beta	**27,200/5 = 5,440**	**27,200/62 = 438**	**440,000/ 27200 = 16.18**		**3,210/272 = 11.80**		

Alpha team productivity is higher than Beta team

Size-oriented Metrics - Normalized

4.3 Software Measurement

Examples of Normalized Size-oriented Metrics:

- Errors per KLOC (Errors/KLOC)
- Defects per KLOC (Defects/KLOC)
- $ per LOC ($/LOC)
- Pages of documentation per KLOC (Pages of doc/KLOC)
- Errors per person-month (Errors/person-month)
- LOC per person-month (LOC/person-month)
- $ per page of documentation ($/Page of doc)

KLOC = thousand lines of code

Function-oriented Metrics

4.3 Software Measurement

Function-oriented Metrics

- **Measures the <u>functionality</u> delivered by the software system** as a normalized value.
- This so-called “functionality” is derived indirectly using Function Point (FP) calculation.

Function-oriented Metrics calculate:

- Function Point (FP)

FP = Count Total x [0.65 + 0.01 ΣF_i]

Count total is the sum of all FP entries

Fi are “complexity adjustment values” where i = 1 to 14

Function-oriented Metrics

4.3 Software Measurement

Function-oriented Metrics

$$FP = \text{Count Total} \times [0.65 + 0.01\ \Sigma F_i]$$

Count Total is the sum of the **counts** from the following *measurement parameters*:

- **No. of user inputs** – elements in the system which require an input from the user (e.g. click on save, print, etc)
- **No. of user outputs** – elements in the system which produce an output (e.g. report, screen, error message, etc)
- **No. of user inquiries** – HELP, Search/Find request
- **No. of files** – database, files, etc
- **No. of external interfaces** - e.g. software communicates with a device (counted as 1 external interface)

Function-oriented Metrics

Computing the Count Total for

To be decided by organization (subjective)

Measurement parameter	count		Weighting factor simple	average	complex		
No. of user inputs		x	3	4	6	=	
No. of user outputs		x	4	5	7	=	
No. of user inquiries		x	3	4	6	=	
No. of files		x	7	10	15	=	
No. of external interfaces		x	5	7	10	=	
					Count total		

Function-oriented Metrics

FP = Count Total x [0.65 + 0.01 ΣF_i]

*Responses are based on the following scale:
0 – no influence
1 – incidental
2 – moderate
3 – average
4 – significant
5 – essential

Complexity adjustment values are obtained from responses* to the following questions:

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?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Function-oriented Metrics

Exercise: Function Point calculation

Measurement Parameter	Count	Weighting factor
Number of inputs	2	Complex (5)
Number of outputs	4	Simple (2)
Number of inquiries	7	Average (4)
Number of files	3	Average (7)
Number of external interfaces	2	Complex (8)
$\Sigma F_i = 60$		

Formulae: FP = **Count Total** x $[0.65 + 0.01\ \Sigma F_i]$

Function-oriented Metrics

Exercise: Calculate **Function Points**

Measurement Parameter	Count	Weighting factor	
Number of inputs	2	Complex (5)	**10**
Number of outputs	4	Simple (2)	**8**
Number of inquiries	7	Average (4)	**28**
Number of files	3	Complex (7)	**21**
Number of external interfaces	2	Complex (8)	**16**
$\Sigma F_i = 60$			**CT = 83**

Formulae: FP = Count Total x $[0.65 + 0.01\ \Sigma F_i]$

= 103.75

Function-oriented Metrics

4.3 Software Measurement

Examples of normalized function-oriented metrics:

- **Errors/ FP**
- **Defects/ FP**
- **$/ FP**
- **Pages doc/ FP**
- **FP/ person-month**

Function-oriented Metrics

Exercise

FP(A) = 67 x [0.65+(0.01 x 60)] = 83.75

FP(B) = 123 x [0.65+(0.01 x 60)] = 153.75

Measurement Parameter	Weighting factor	Project A's Count	Project B's Count
Number of inputs	Simple(2)	2	5
Number of outputs	Simple(2)	4	12
Number of inquiries	Average(4)	7	9
Number of files	Average(5)	3	7
Number of external interfaces	Complex(6)	2	3
$\Sigma F_i = 60$			

- Pages of doc per FP for P(A) = 120/83.75=1.43
- Pages of doc per FP for P(B) = 190/153.75 =1.24

P(A) has more pages of doc per FP. Means P(A) degree of <u>maintainability</u> is higher than P(B)

a. Calculate function point (FP) for both project A and B.

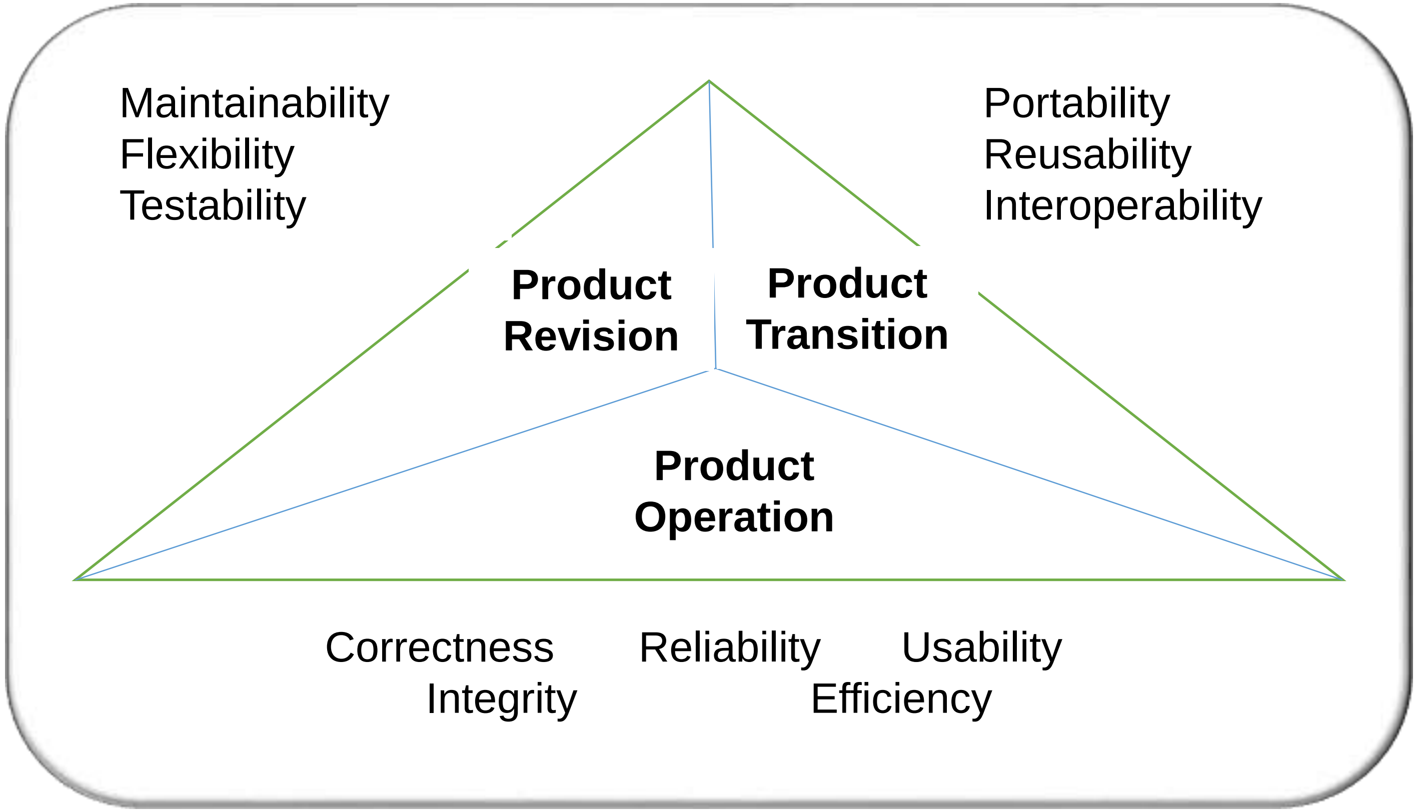
b. <u>120</u> pages of documentation is found for Project A while <u>190</u> pages for Project B. Evaluate which project has higher maintainability by using FP in your normalized measurement.

4.4 Metrics for Software Quality Attributes

- McCall’s Software Quality Factors
- Metrics for Measuring Software Quality
- Challenges in Measuring Software
- Characteristics of Good Software Metrics

McCall's Software Quality Factors

4.4 Metrics for Software Quality Attributes



3 software quality factors categories:

- Product operation (using it)
- Product revision (changing it)
- Product transition (modifying it to work in a different environment; i.e., "porting" it)

Metrics for Measuring Software Quality

4.4 Metrics for Software Quality Attributes

- **What <u>metrics</u> can we use for the following software quality attributes?**
  a. Correctness
  b. Maintainability
  c. Usability

Metrics for Measuring Software Quality

4.4 Metrics for Software Quality Attributes

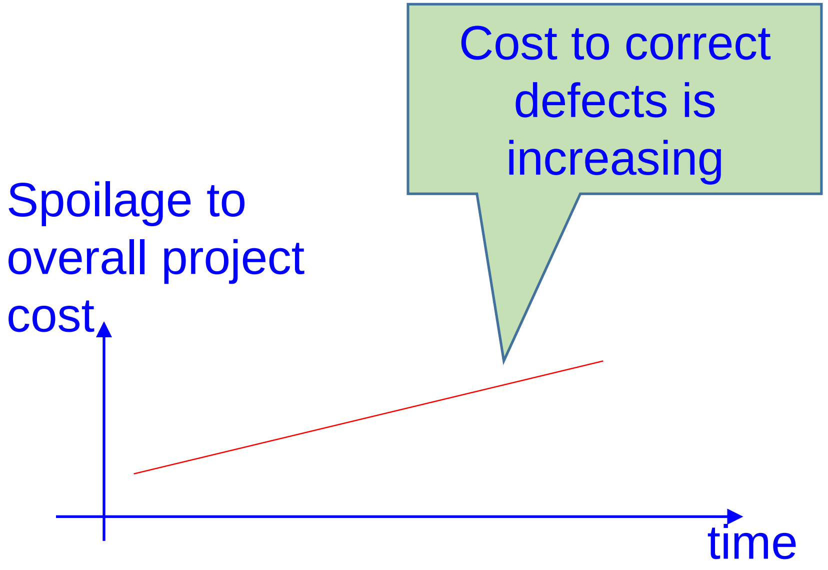
a. Correctness

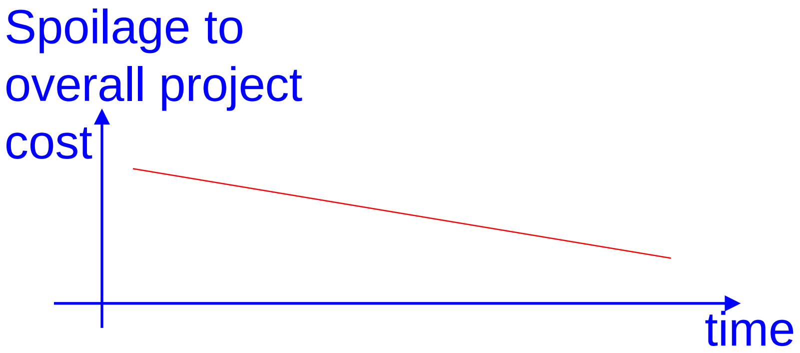
- **The degree to which the software performs its required function**
- Common **measures**:
  - Defects per KLOC
  - Defects over a standard period of time. (i.e. one year)

Note*: *Defect refers to a verified lack of conformance to requirements.

b. Maintainability

- The ease with which a program can be corrected if an error is encountered, adapted if its environment changes, or enhanced if the customer desires a change in requirements.
- No direct way to measure, so must use indirect **measures**:
    - Mean-time-to-change (MTTC)
        - **The time it takes to:**
            - **analyse the change request,**
            - **design an appropriate modification,**
            - **implement the change,**
            - **test it, and**
            - **distribute the change to all users.**
        - **Highly maintainable programs will have a lower MTTC.**
    - Spoilage
        - **The cost to correct defects encountered after the software has been released to its end-users.**
        - When the ratio of spoilage to overall project cost is plotted as a function of time, a manager can determine whether the overall maintainability of software produced by a software development organization is improving.





Metrics for Measuring Software Quality

4.4 Metrics for Software Quality Attributes

c. Usability

- How easy it is to use the system
- Can be measured in terms of these characteristics:
  - **The time required to learn how to perform a task for the first time using the system** (e.g. record a video using Gmeet)
  - **The time required to become moderately efficient in the use of the system** (e.g. time needed to become moderately efficient in using all features in GC
  - **The net increase in productivity, measured against the old process or system, measured after a user has gained moderate efficiency**
  - **A subjective measure of user attitude towards the system (using a questionnaire)**

Challenges in Measuring Software

4.4 Metrics for Software Quality Attributes

- **Measurement is too complex**
- Too esoteric that **very few professionals could understand**
- Violate the basic intuitive notions of what high-quality software really is
- Many researchers attempted to develop a single metric that provides a comprehensive measure of software complexity
- Derived metrics might not be useful or suitable without realizing it. In other words, metrics might not prove anything
- **Collecting measures is time consuming**
- **Difficult to determine what to measure and evaluating measures that are collected**

Characteristics of Good Metrics

4.4 Metrics for Software Quality Attributes

- Consistent in its use of units and dimensions
  - The mathematical computation of the metric should use measures that do not lead to bizarre combinations of units.
  - E.g. use size and/or FP throughout all projects
- Programming language independent
  - Metrics should be based on the analysis model, the design model, or the structure of the program itself.
  - E.g. LOC is programming language dependent
- Simple and computable
  - Relatively easy to learn how to derive the metric
  - Its computation should not demand inordinate effort or time

Characteristics of Good Metrics

4.4 Metrics for Software Quality Attributes

- Empirically and intuitively persuasive
  - Metrics should match the practitioner’s notions about the product attribute under consideration
- Consistent and objective
  - Always yield results that are unambiguous
- An effective mechanism for high-quality feedback
  - Should motivate team for software development improvement
  - The metrics should lead to a higher-quality end product

Summary

4.1 Introduction

4.2 Software Metrics

4.3 Software Measurement

4.4 Metrics for Software Quality Attributes