# CSE4006 Software Engineering

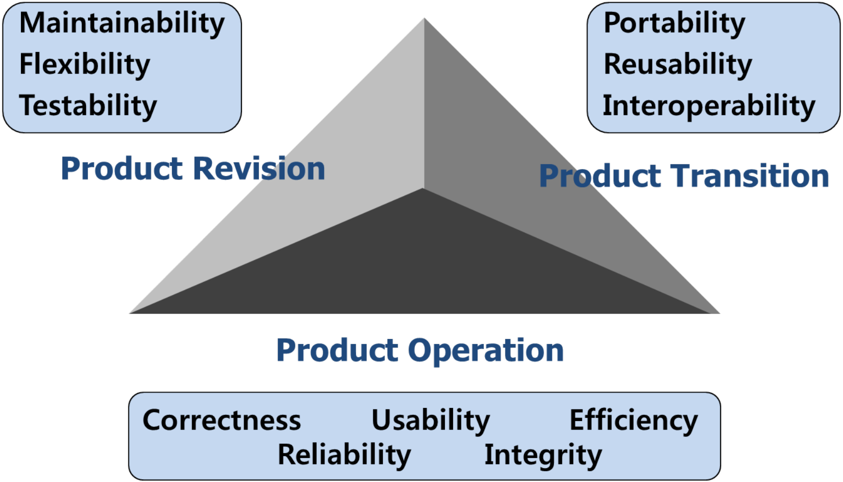
# 12. Product Metrics

Scott Uk-Jin Lee

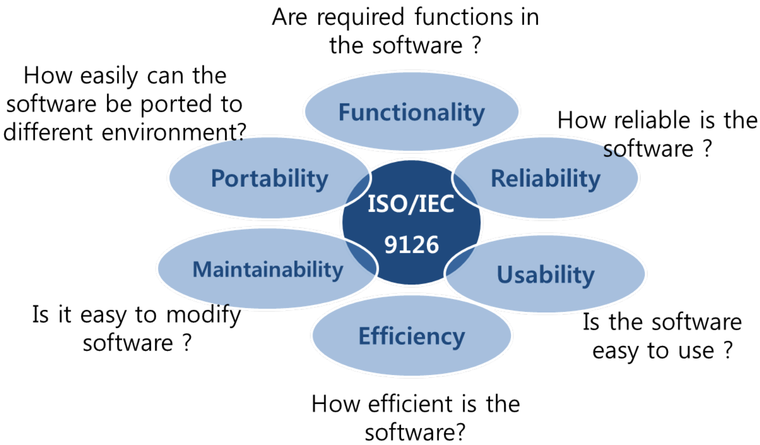
Department of Computer Science and Engineering
Hanyang University ERICA Campus

$1^{st}$ Semester 2015

# 1 McCall's Triangle of Quality (1970)



# 1.1 ISO 9126 Software Quality Characteristics




lab(se);

# 1.2 Measurement of Quality Characteristics

- indirect measurement vs. direct measurement
- Product metrics vs. Process metrics
  - Product perspective: program size, complexity, data size, functionality, reliability,consistency, maintainability ...
  - Process perspective: development time, cost, and progress
- Metrics for prediction (estimation)

lab(se);

# 2 Measures, Metrics, indicators

- A SW engineer collects **measures** and develops **metrics** to obtain **indicators**
  - A **measures** provides a **quantitative** indication of the extent, amount, dimension, capacity, or size of some attributes of a product or process
  - The IEEE defines a **metics** as "a quantitative measure of the degree to which a system, component, or process possesses a given **attribute**." - IEEE Standard Glossary of SE Terminology
  - An **indicator** is a metric or combination of metrics that provide **insight** into the software process, project, or the product itself

**e.g.,** Scarlett Johansson

- measure: height $=$ 162cm,
  weight $=$ 51kg
- metric: BMI metric $=$ 19.8
  (weight(kg)/(height(m) × height(m)))
- indicator: normal

| BMI | Indicator |
|---|---|
| ≤ 18.5 | underweight |
| 18.5 – 24.9 | normal |
| 25.0 – 29.9 | overweight |
| 30.0 ≤ | Obesity |

(se);

# 2.1 Measurement Principles

- **Objectives** of measurement should be established before data collection begins

  **e.g.,** measuring number of words in a C file might be useless

- Each technical metric should be defined in an **unambiguous** manner

  **e.g.,** For measuring a total line number of a C program

  - include comments? include empty line?

- Metrics should be derived based on a **theory** that is valid for the domain of application
  - metrics for design should draw upon **basic design concepts and principles** and attempt to provide an **indication** of the presence of a desirable attribute
  - metrics should be **tailored** to best accommodate specific products and processes

# 2.2 Measurement Process

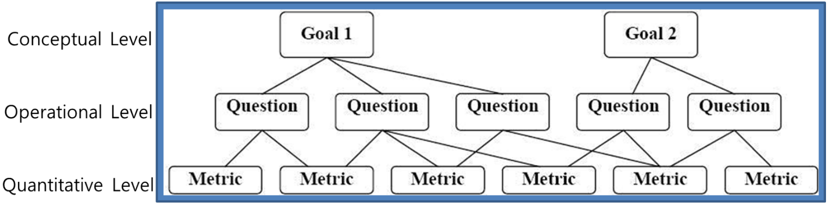
- Formulation
  - derivation of software measures and metrics appropriate for the representation of the software being considered
- Collection
  - mechanism used to accumulate data required to derive the formulated metrics
- Analysis
  - computation of metrics and the application of mathematical tools
- Interpretation
  - evaluation of metrics results in an effort to gain insight into the quality of the representation
- Feedback
  - recommendations derived from the interpretation of product metrics transmitted to the software team

# 2.2 Measurement Process Examples

- Formulation
  - check wether a given software is hot-spotted (has intensive loops)
- Collection
  - instrument a source program/binary to count how many times a given statement is executed in one second
- Analysis
  - use Exel/Matlab to get average number of execution for each statements
- Interpretation
  - if there exists statements that were executed more than $10^8$ times on a 3Ghz Machine, then the program is hot-spotted
- Feedback
  - try to optimize those hot-spotted statements or check if those hot-spotted statements have any logical flaw

# 2.3 Goal-Oriented Software Measurement

**GQM** (Goal/Question/Metric) Paradigm




- establish an explicit measurement **goal**
- define a set of **questions** that must be answered to achieve the goal
- identify well-formulated **metrics** that help to answer these questions

lab(se);

# 2.3 Goal-Oriented Software Measurement

- Goal definition template
  - **Analyze** {the name of activity or attribute to be measured}
  - **for the purpose of** {the overall objective of the analysis}
  - **with respect to**
    {the aspect of the activity of attribute that is considered}
  - **from the viewpoint of**
    {the people who have an interest in the measurement}
  - **in the context of**
    {the environment in which the measurement takes place}
- GQM exmaple
  - Goal : Analyze final product to characterize it with respect to various defect class from the viewpoint of of the organization
  - Question : What is the error distributed by phase of entry
  - Metric: Number of Requirements Errors, Number of Design Errors, ...

# 2.4 Metrics Attributes

- Simple and computable
  - it should be relatively easy to learn how to derive the metric, and its computation should not demand inordinate effort/time
- Empirically and intuitively persuasive
  - the metric should satisfy the engineer's intuitive notions about the product attribute under consideration
- Consistent and objective
  - the metric should always yield results that are unambiguous
- 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 unit
- Programming language independent
  - metrics should be based on the analysis model, the design model, or the structure of the program itself
- Effective mechanism for quality feedback
  - the metric should provide a software engineer with information that can lead to a higher quality end product

# 2.5 Collection and Analysis Principles

- **Automate** data collection and analysis
- Apply statistical techniques to establish relationship between internal product attributes and external quality characteristics
- Establish interpretative guidelines and recommendations for each metric

# 3 Metrics for the Analysis Model

- **Function-based metrics**
  - use the function point (FP) as a normalizing factor or as a measure of the "size" of the specification
- **Specification metrics**
  - used as an indication of quality by measuring number of requirements by type

# 3.1 Function-based metrics

## function point metric (FP)

- first proposed by Albrecht in 1979
- used effectively as a means for measuring the functionality delivered by a system
- based on countable (direct) measures of software's information domain and assessments of software complexity
- Definition of information domain values :
  - 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)

# 3.1 Function Points

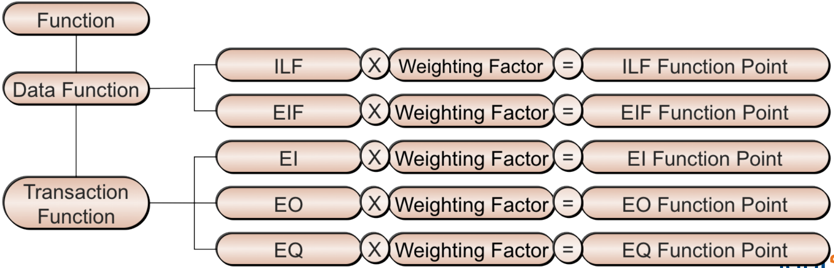
| Information Domain Value | Count | | Weighting Factor | | | Result |
|---|---|---|---|---|---|---|
| | | | Simple | Average | Complex | |
| External Inputs (EIs) | | × | 3 | 4 | 6 | = |
| External Outputs (EOs) | | × | 4 | 5 | 7 | = |
| External Inquiries (EQs) | | × | 3 | 4 | 6 | = |
| Internal Logical File (ILFs) | | × | 7 | 10 | 15 | = |
| External Interface File (EIFs) | | × | 5 | 7 | 10 | = |
| Count Total | | | | | | |

# 3.1 Function Point

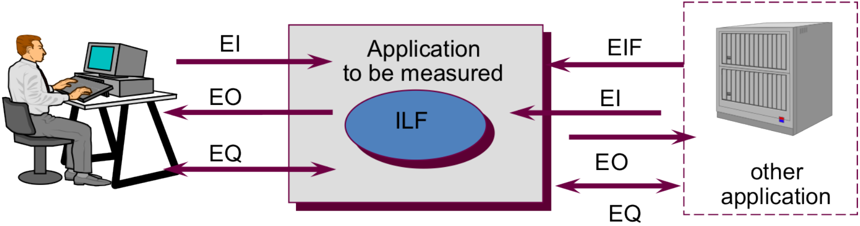
- LOC per 1 FP
  - Assembler 320, C 150, Pascal 91, Ada 71, APL 32
- Total LOC = FP * LOC per 1 point for the desired language
  - in case of C language : 1FP = 150 LOC
  - **148 FP** = 148 * 150 LOC = 22,200 LOC = **22.2KLOC**
- Development effort = total LOC / productivity (LOC/MM)
- Function point calculation method defined by International Function Point User Group (IFPUG) are used

# 3.2 IFPUG Function Point

- IFPUG : International Function Point User Group
  - After deriving Data function and Transaction function, apply complexity weighting factor to calculate FP
  - Data function is logical table where information exists
  - Transaction function is function provided for data processing (register, modify, delete, query, output etc)



# 3.2 IFPUG Function Point




- Complexity Weighting Factor
  - determined by the number of data element types (DET) and record element types (RET) in ILF and EIF
  - determined by the number of file type referenced (FTR) in EI/EO/EQ

# 3.3 Function Points Calculation

- Step 1. calculate unadjusted FP

| Information Domain Value | Count | | Weighting Factor | | | Result |
|---|---|---|---|---|---|---|
| | | | Simple | Average | Complex | |
| External Inputs (EIs) | | × | 3 | 4 | 6 | = |
| External Outputs (EOs) | | × | 4 | 5 | 7 | = |
| External Inquiries (EQs) | | × | 3 | 4 | 6 | = |
| Internal Logical File (ILFs) | | × | 7 | 10 | 15 | = |
| External Interface File (EIFs) | | × | 5 | 7 | 10 | = |
| Count Total → | | | | | | |

- Step 2. consider 14 general system characteristics and calculate Total Degree of Influence (TDI)
  - refer to the previous slide
- Step 3. calculate final FP
  - **FP = UFP x (0.65 + 0.01 x TDI)**

lab(se);

# 3.3 Function Points Calculation

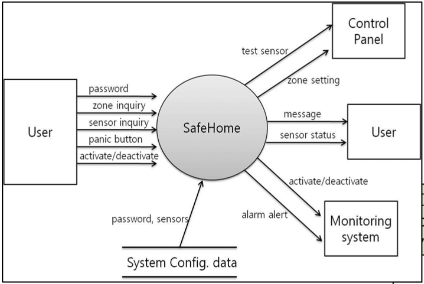
## Total Degree of Influence (TDI) Calculation

| General System Characteristics | DI | General System Characteristics | DI |
|---|---|---|---|
| Data Communication | 4 | Online Update | 3 |
| Distributed Data Processing | 0 | Complex Processing | 0 |
| Performance | 0 | Reusability | 3 |
| Heavily Used Configuration | 0 | Installation Ease | 0 |
| Transaction Role | 0 | Operational Ease | 3 |
| Online Data Entry | 5 | Multiple Sites | 2 |
| End-user Efficiency | 3 | Facilitate Change | 2 |

❖ Degree of Influence (DI)

Depending on how 14 characteristics affect system development rate in the scale of 0~5

# 3.3 Function Points Calculation




| Info. Domain Value | Count | Weighting Factor | | | Result |
|---|---|---|---|---|---|
| | | Simple | Average | Complex | |
| EIs | **3** × | 3 | 4 | 6 | = **9** |
| EOs | **2** × | 4 | 5 | 7 | = **8** |
| EQs | **2** × | 3 | 4 | 6 | = **6** |
| ILFs | **1** × | 7 | 10 | 15 | = **7** |
| EIFs | **4** × | 5 | 7 | 10 | = **20** |
| **Count Total(UFP)** | | | | | **50** |

# 3.3 Function Points Calculation

- Assume that we have calculated TDI = 46 (all middle)

$$FP = UFP \times (0.65 + 0.01 \times TDI)$$
$$= 50 \times (0.65 + 0.01 \times 46) = 56$$

- OO language 1FP = 60 LOC,

  12 FP = 1 man-month of effort

lab(se);

# 4 Architecture Design Metrics

- Architectural design metrics
  - Structural complexity = g(fan-out)
  - Data complexity = f(input & output variables, fan-out)
  - System Complexity = h(structural & data complexity)
- HK Metric
  - architectural complexity as a function of fan-in and fan-out
- Morphology metric
  - function of a number of modules and the number of interfaces between modules

# 4.1 Object-Oriented Design Metrics

In 1997, Whitmire describes nine distinct and measurable characteristics of an object-oriented design

- Size
  - population, volume, length, and functionality
- Complexity
  - how classes of an OO design are interrelated to one another
- Coupling
  - physical connection between elements of the OO design
- Sufficiency
  - the degree to which an abstraction possesses the features required of it, or the degree to which a design component possesses features in its abstraction

# 4.1 Object-Oriented Design Metrics

- Completeness
  - indirect implication about the degree to which the abstraction or design component can be reused
- Cohesion
  - degree to which all operations working together to achieve a single, well-defined purpose
- Primitiveness
  - applied to both operations and classes, the degree to which an operation is atomic
- Similarity
  - degree to which two or more classes are similar in terms of their structure, function, behavior, or purpose
- Volatility
  - measures the likelihood that a change will occur

# 4.2 Class-Oriented Metrics

- CK Metrics : proposed by Chidamber and Kemerer in 1994
  - weighted methods per class (WMC)
  - depth of the inheritance tree (DIT)
  - number of children (NOC)
  - coupling between object classes (CBO)
  - response for a class (RFC)
  - lack of cohesion in methods (LCOM)

- LK Metrics : proposed by Lorenz and Kidd in 1994
  - class size (= total number of operations)
  - number of operations overridden by a subclass
  - number of operations added by a subclass
  - specialization index (current class level)

# 4.2 Class-Oriented Metrics

- MOOD Metrics : proposed by Fernando Brito in 1998
- original MOOD metrics consists of 6 metrics & MOOD2 metrics were added later
  - Method / Attribute Hiding Factor (MHF / AHF)
  - Method / Attribute Inheritance Factor (MIF / AIF)
  - Coupling Factor (CF)
  - Polymorphism Factor (PF)

# 4.3 Operation-Oriented Metrics

- proposed by Lorenz and Kidd in 1994
    - average operation size
    - operation complexity
    - average number of parameter per operation

# 5 Code Metrics (Limitation of LOC)

- Line Of Code (LOC)
  - not a size measurement on the customer's requirement
  - have consistency issues to be a measure because languages and tools used as well as programmers causes a large differences on productivity
  - size measurement is impossible in steps prior to code generation
- **Function Point & Halstead's Software Science**
  - in 2004, FP became domestic SW business cost standard
    - In 2009 2010, LOC and man month was abolished as SW development cost calculating methods
  - Halstead's Software Science is utilized as a measure for maintenance effort estimation

lab(se);

# 5 Code Metrics

- **Halstead's Software Science**
  - proposed various predictable metrics using the number of operators and operands in a well-structured program
  - n1 - type of operators used in a program
  - n2 - type of operands used in a program
  - N1 - total number of operators used in a program
  - N2 - total number of operands used in a program
  - **Estimated program length : N**

    $$N = n1 \log_2 n1 + n2 \log_2 n2$$

  - **potential volume : V**

    $$V = N \log_2 (n1 + n2)$$

# 5 Code Metrics

```
      SUBROUTINE SORT(X,N)
      DIMENSION X(N)
      IF (N.LT.2) RETURN
      DO  20 I = 2,N
         DO 10 J = 1,I
         IF (X(I) .GE. X(J)) GO TO 10
            SAVE = X(I)
            X(I) = X(J)
            X(J) = SAVE
10    CONTINUE
20    CONTINUE
      RETURN
      END
```

| | operator | number |
|---|---|---|
| | 1 End of statement | 7 |
| | 2 Array subscript | 6 |
| | 3 = | 5 |
| | 4 IF ( ) | 2 |
| | 5 DO | 2 |
| | 6 , | 2 |
| | 7 End of program | 1 |
| | 8 .LT. | 1 |
| | 9 .GE. | 1 |
| n1=10 | GO TO 10 | 1 |
| | | 28 = N1 |

| | operand | | number |
|---|---|---|---|
| | 1 | X | 6 |
| | 2 | I | 5 |
| | 3 | J | 4 |
| | 4 | N | 2 |
| | 5 | 2 | 2 |
| | 6 | SAVE | 2 |
| n2=7 | | 1 | 1 |
| | | | 22 = N2 |

- Difficulty Level (D) = (n1 / 2)*(N2 / n2)
  - proportional to the number of unique operators in a program
- Program Level (L) = 1 / D
  - shows low level programs tend to have more errors than high level program
- **Effort (E)**
  - the extent of effort required to understand or implement a program is proportional to the volume (V) and difficulty (D)

$$E = V * D = \frac{V}{L} = \frac{(n_1 * N_2 (N_1 + N_2) * \log_2 (n_1 + n_2))}{2 * n_2}$$

lab(se);

# 5 Code Metrics

- Typical application of **Halstead's Software Science**
  - operator : reserved word or symbols that are fixed
  - operand : everything else (variable names, function names, numeric constants, etc)
  - comments are not counted

# 5 Code Metrics

```
int f=1, n=7;
for (int i=1; i<=n; i+=1)
  f*=i;
```

1. type of operators used in a program, n1 = 10
   $int \quad = \quad , \quad ; \quad for \quad ( \quad <= \quad += \quad ) \quad *=$
2. type of operands used in a program, n2 = 5
   $f \quad 1 \quad n \quad 7 \quad i$
3. total number of operators used in a program, N1 = 16
   $int \quad = \quad , \quad = \quad ; \quad for \quad ( \quad int \quad = \quad ; \quad <= \quad ; \quad += \quad ) \quad *= \quad ;$
4. total number of operands used in a program, N2 = 12
   $f \quad 1 \quad n \quad 7 \quad i \quad 1 \quad i \quad n \quad i \quad 1 \quad f \quad i$

lab(se);

# 6 Testing Metrics

- Testing Effort can be estimated using Halstead's measures
- In 1994, Binder suggests a broad array of design metrics that have a direct influence on the "testability" of an OO system
  - Lack of cohesion in methods (LCOM)
  - Percent public and protected (PAP)
  - Public access to data members (PAD)
  - Number of root classes (NOR)
  - Fan-in (FIN)
  - number of children (NOC) and depth of the inheritance tree (DIT)

# 7 Maintenance Metrics

- Software Maturity Index (SMI)
  - IEEE standard 982.1-1998
  - indication of the stability of a software product
    (based on changes that occur for each product release)
  - SMI = $[M_t - (F_a + F_c + F_d)]/M_t$
    - $M_t$ = No. of modules in current release
    - $F_c$ = No. of modules in current release that have changed
    - $F_a$ = No. of modules in current release that have added
    - $F_d$ = No. of modules in current release that have deleted
  - as SMI approach 1.0, the product begins to stabilize