Chapter 15
Product Metrics

Moonzoo Kim
CS Division of EECS Dept.
KAIST
moonzoo@cs.kaist.ac.kr
http://pswlab.kaist.ac.kr/courses/cs550-07

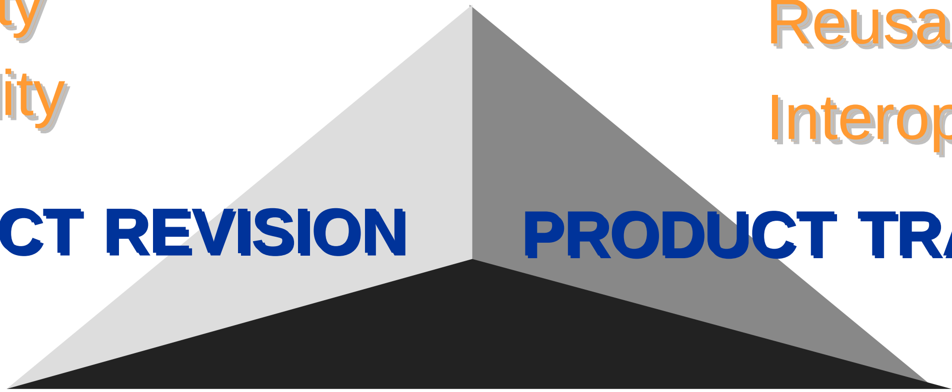
Overview of Ch15. Product Metrics

- 15.1 Software Quality
- 15.2 A Framework for Product Metrics
- 15.3 Metrics for the Analysis Model
  - Function point metrics
- 15.4 Metrics for the Design Model
  - Architectural design metrics
  - Metrics for OO design
  - Class-oriented metrics
  - Component-level design metrics
  - Operation oriented metrics
- 15.5 Metrics for Source Code
- 15.6 Metrics for Testing
- 15.7 Metrics for Maintenance

McCall's Triangle of Quality (1970s)

Maintainability
Flexibility
Testability

Portability
Reusability
Interoperability



PRODUCT REVISION

PRODUCT TRANSITION

PRODUCT OPERATION

Correctness
Usability
Efficiency
Reliability
Integrity

SW built to conform to these factors will exhibit high quality, even if there are dramatic changes in technology.

Measures, Metrics and Indicators

- A SW engineer collects measures and develops metrics so that indicators will be obtained
  - 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."
    - IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990)
  - An *indicator* is a metric or combination of metrics that provide insight into the software process, a software project, or the product itself

Measurement Principles

- The objectives of measurement should be established before data collection begins
- Each technical metric should be defined in an unambiguous manner
- 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

Measurement Process

- *Formulation*.

  The derivation of software measures and metrics appropriate for the representation of the software that is being considered.

- *Collection*

  The mechanism used to accumulate data required to derive the formulated metrics.

- *Analysis*.

  The computation of metrics and the application of mathematical tools.

- *Interpretation*.

  The 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.

Goal-Oriented Software Measurement

- The 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.
- 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 or 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}.

Ex> Goal definition for SafeHome

- **Analyze** the Safehome SW architecture **for the purpose of** evaluating architectural components **with respect to** the ability to make Safehome more extensible **from the viewpoint of** the SW engineers performing the work **in the context of** produce enhancement over the next 3 years
- Questions
  - Q1: Are architectural components characterized in a manner that compartmentalizes function and related data?
    - Answer: 0 … 10
  - Q2: Is the complexity of each component within bounds that will facilitate modification and extension?
    - Answer: 0 … 1

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 or 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.
- *an effective mechanism for quality feedback.*

  That is, the metric should provide a software engineer with information that can lead to a higher quality end product

Collection and Analysis Principles

- Whenever possible, data collection and analysis should be automated
- Valid statistical techniques should be applied to establish relationship between internal product attributes and external quality characteristics
- Interpretative guidelines and recommendations should be established for each metric

Function-Based Metrics

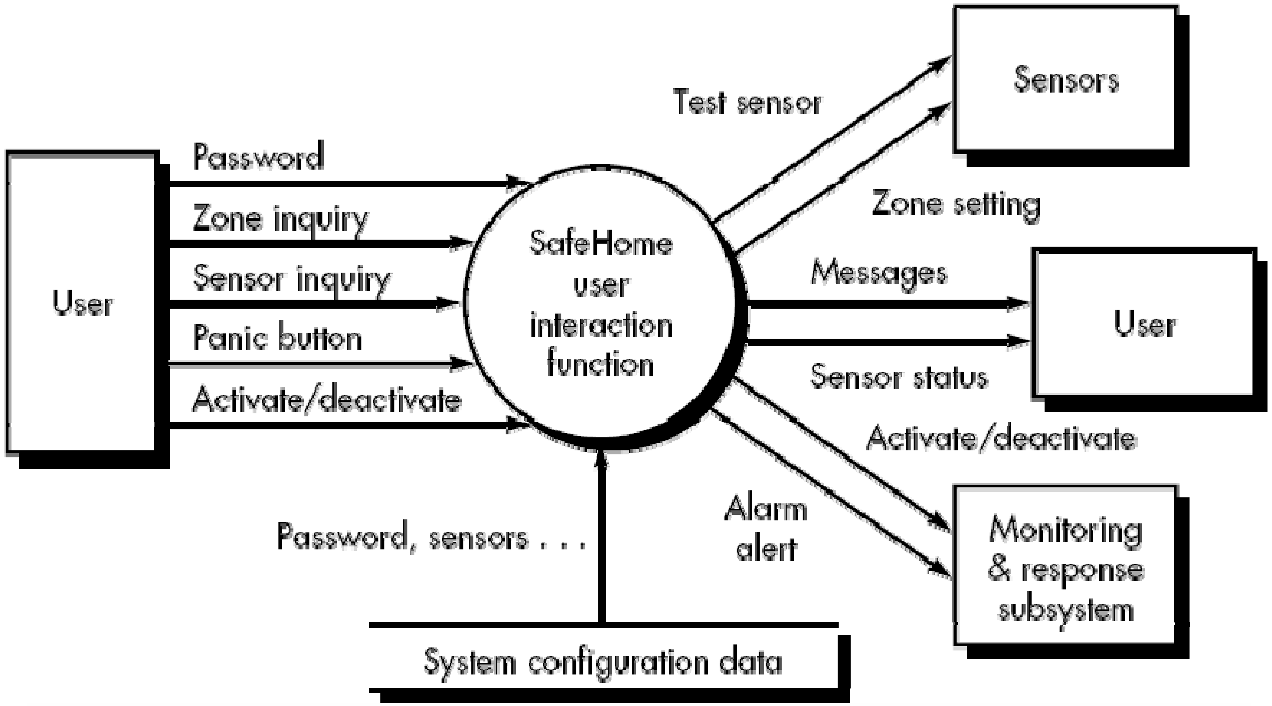
- The *function point metric* (FP), first proposed by Albrecht [ALB79], can be used effectively as a means for measuring the functionality delivered by a system.
- Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments 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

Information Domain Value	Count		Weighting factor simple	average	complex		
External Inputs (EIs)	☐	3	3	4	6	=	☐
External Outputs (EOs)	☐	3	4	5	7	=	☐
External Inquiries (EQs)	☐	3	3	4	6	=	☐
Internal Logical Files (ILFs)	☐	3	7	10	15	=	☐
External Interface Files (EIFs)	☐	3	5	7	10	=	☐
Count total	→						☐

$FP = \text{count total} \times (0.65 + 0.01 \times \sum(F_i))$

where Fi's are value adjustment factors based on responses to the 14 questions (473 pg of SEPA)



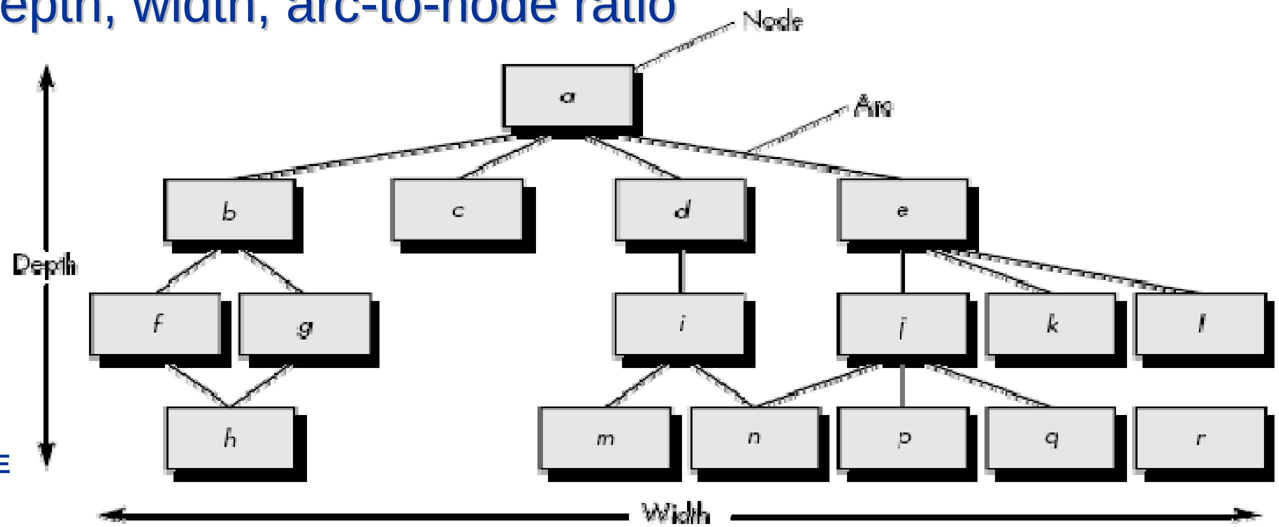
Measurement parameter	Count		Simple	Average	Complex		
			Weighting Factor				
Number of user inputs	3	×	3	4	6	=	9
Number of user outputs	2	×	4	5	7	=	8
Number of user inquiries	2	×	3	4	6	=	6
Number of files	1	×	7	10	15	=	7
Number of external interfaces	4	×	5	7	10	=	20
Count total						→	50

Usage of Function Points

- Assume that
  - past data indicates that one FP translates into 60 lines of code
  - 12 FPs are produced for each person-month of effort
  - Past projects have found an average of 3 errors per FP during analysis and design reviews
  - 4 errors per FP during unit and integration testing
- Suppose that Safehome has 56 FPs
  - $\sum(F_i) = 46$
- These data can help SW engineers assess the completeness of their review and testing activities

Architectural Design Metrics (black box)

- Architectural design metrics
  - Structural complexity of a module m= (# of fan-out of module m)^2
  - Data complexity = (# of input & output variables)/ (fan-out+1)
  - System complexity = structural complexity + data complexity)
- Morphology metrics: a function of the number of modules and the number of interfaces between modules
  - Size, depth, width, arc-to-node ratio



Metrics for OO Design-I

- Whitmire [WHI97] describes nine distinct and measurable characteristics of an OO design:
  - Size
    - Size is defined in terms of four views: population, volume, length, and functionality
  - Complexity
    - How classes of an OO design are interrelated to one another
  - Coupling
    - The physical connections 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, from the point of view of the current application.”
  - Completeness
    - An indirect implication about the degree to which the abstraction or design component can be reused

Metrics for OO Design-II

- Cohesion
  - The 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
  - The 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

Distinguishing Characteristics

Berard [BER95] argues that the following characteristics require that special OO metrics be developed:

- Localization

  the way in which information is concentrated in a program

- Encapsulation

  the packaging of data and processing

- Information hiding

  the way in which information about operational details is hidden by a secure interface

- Inheritance

  the manner in which the responsibilities of one class are propagated to another

- Abstraction

  the mechanism that allows a design to focus on essential details

Class-Oriented Metrics

Proposed by Chidamber and Kemerer (CK metrics):

- weighted methods per class

  $\sum(m_i)$ where m_i is a normalized complexity for method i
- depth of the inheritance tree
- number of children
- coupling between object classes
- response for a class
- lack of cohesion in methods

Applying CK Metrics (pg483-484)

- **The scene:**
  - Vinod's cubicle.
- **The players:**
  - Vinod, Jamie, Shakira, Ed

    members of the *SafeHome* software engineering team, who are continuing work on component-level design and test case design.
- **The conversation:**
- **Vinod:** Did you guys get a chance to read the description of the CK metrics suite I sent you on Wednesday and make those measurements?
- **Shakira:** Wasn't too complicated. I went back to my UML class and sequence diagrams, like you suggested, and got rough counts for DIT, RFC, and LCOM. I couldn't find the CRC model, so I didn't count CBO.
- **Jamie (smiling):** You couldn't find the CRC model because I had it.
- **Shakira:** That's what I love about this team, superb communication.
- **Vinod:** I did my counts . . . did you guys develop numbers for the CK metrics?

- (Jamie and Ed nod in the affirmative.)
- **Jamie:** Since I had the CRC cards, I took a look at CBO, and it looked pretty uniform across most of the classes. There was one exception, which I noted.
- **Ed:** There are a few classes where RFC is pretty high, compared with the averages . . . maybe we should take a look at simplifying them.
- **Jamie:** Maybe yes, maybe no. I'm still concerned about time, and I don't want to fix stuff that isn't really broken.
- **Vinod:** I agree with that. Maybe we should look for classes that have bad numbers in at least two or more of the CK metrics. Kind of two strikes and you're modified.
- **Shakira (looking over Ed's list of classes with high RFC):** Look, see this class? It's got a high LCOM m well as a high RFC. Two strikes?
- **Vinod:** Yeah I think so . . . it'll be difficult to implement because of complexity and difficult to test for the same reason. Probably worth designing two separate classes to achieve the same behavior.
- **Jamie:** You think modifying it'll save us time?
- **Vinod:** Over the long haul, yes.

Class-Oriented Metrics

The MOOD Metrics Suite

- Method inheritance factor

  MIF = $\sum \mathbf{M_i(C_i)} / \sum \mathbf{M_a(C_i)}$

- Coupling factor

  CF = $\sum \sum \mathbf{is\_client(C_i,C_j)} / (\mathbf{T_c^2 - T_c})$

Class-Oriented Metrics

Proposed by Lorenz and Kidd [LOR94]:

- class size
- number of operations overridden by a subclass
- number of operations added by a subclass

Component-Level Design Metrics

- Cohesion metrics

  a function of data objects and the locus of their definition
- Coupling metrics

  a function of input and output parameters, global variables, and modules called
- Complexity metrics

  hundreds have been proposed (e.g., cyclomatic complexity)

Operation-Oriented Metrics

Proposed by Lorenz and Kidd [LOR94]:

- average operation size

  # of messages sent by the operation
- operation complexity
- average number of parameters per operation

Metrics for Testing

- Testing effort can also be estimated using metrics derived from Halstead measures
- Binder [BIN94] 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).

Metrics for Maintenance

- IEEE Std 982.1-1998 Software Maturity index (SMI)
  - $SMI = [M_T - (F_a + F_c + F_d)]/M_T$
    - M_t = # of modules in the current release
    - F_c = # of modules in the current release that have been changed
    - F_a = # of modules in the current release that have been added
    - F_d = # of modules from the preceding release that were deleted in the current release