Introduction to Software Engineering
(CS350)

Lecture 19

Jongmoon Baik

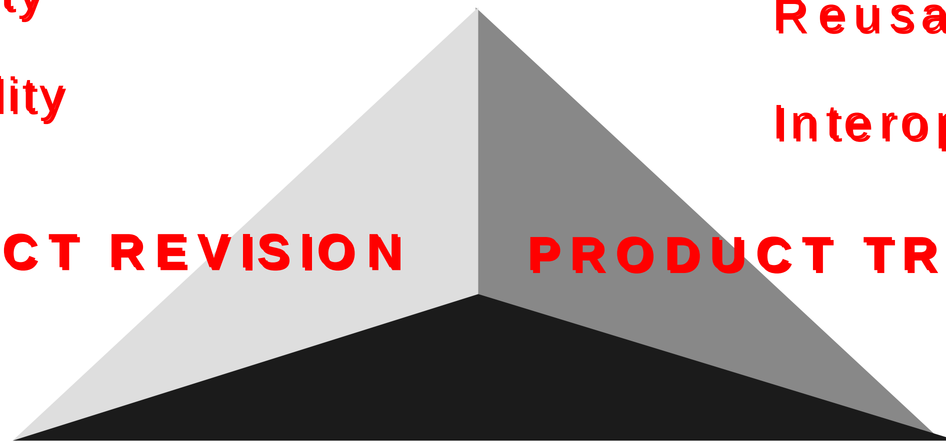
Software M&M – I

Product Metrics

McCall's Triangle of Quality

Maintainability
Flexibility
Testability

Portability
Reusability
Interoperability



PRODUCT REVISION

PRODUCT TRANSITION

PRODUCT OPERATION

Correctness
Usability
Efficiency
Reliability
Integrity

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
  - *Measurement* is the act of determining measure
- 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

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
  - (e.g., metrics for design should draw upon basic design concepts and principles and attempt to provide an indication of the presence of an attribute that is deemed desirable);
- Metrics should be tailored to best accommodate specific products and processes [Bas84]

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

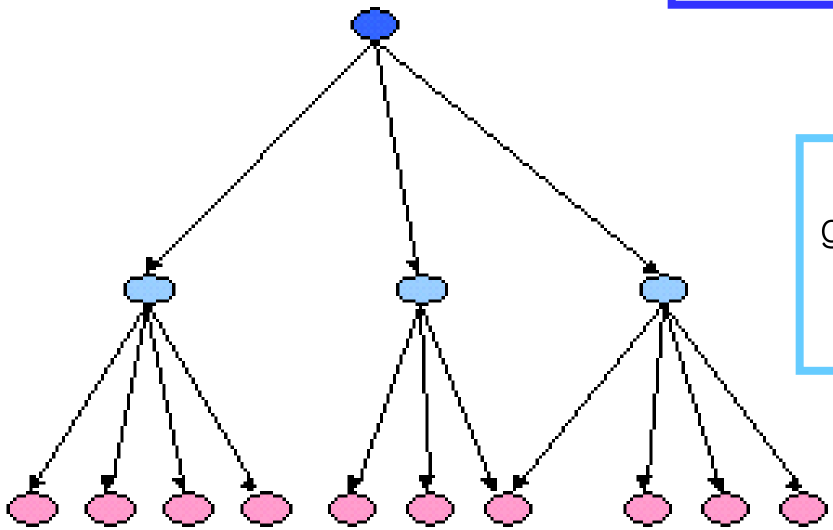
- The Goal/Question/Metric Paradigm
  - (1) establish an explicit measurement *goal* that is specific to the process activity or product characteristic that is to be assessed
  - (2) define a set of *questions* that must be answered in order to achieve the goal, and
  - (3) identify well-formulated *metrics* 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}.

What is GQM?

"A systematic approach for integrating goals to models of the software processes, products and quality perspectives of interest based upon the specific needs of the project and the organization"

[Basili 1984]

Three Steps of GQM



Goal: The major goals of the development project

Questions: Questions derived from goals that must be answered in order to determine if the goals are achieved

Metrics: Measurements that provide the most appropriate information for answering the questions

GQM Example

Goal	Question	Metrics
Plan	How much does the inspection process cost?	Average effort per KLOC Percentage of the reinspections
	How much calendar time does the inspection process take?	Average effort per KLOC Total KLOC inspected
Monitor & Control	What is the quality of the inspected software?	Average faultes detected per KLOC Average inspection rate Average preparation rate
	To what degree did the staff conform to the procedures	Average inspection rate Average preparation rate Average lines of code inspected
	What is the status of the inspection process?	Total KLOC inspected
Improve	How effective is the inspection process?	Defect removal efficiency Average faults detected per KLOC Average inspection rate Average preparation rate Average lines of code inspected
	What is the productivity of the inspection process?	Average effort per fault detected Average inspection rate Average preparation rate Average lines of code inspected

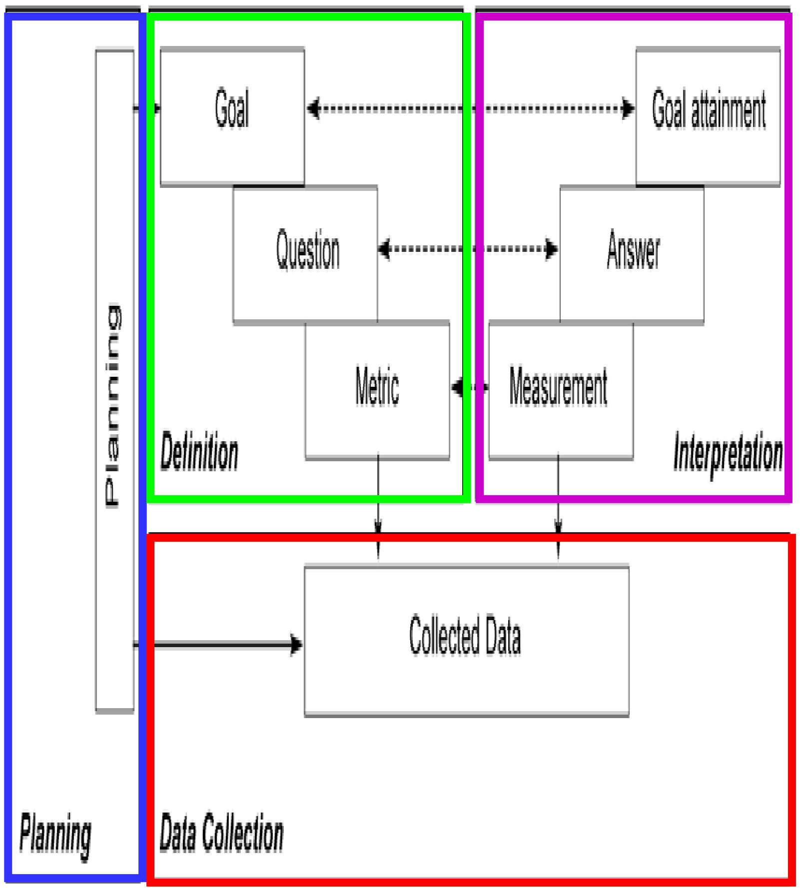
Source: N. Fenton & S. Pfleeger, Software Metrics: A Rigorous & Practical Approach (2nd Ed), PWS Publishing Company, 1997

GQM & Software Process Improvement

- Two major streams of improvement models, methods, and techniques
  - Top-down approaches
    - Which are based on assessment and benchmarking
    - Example: CMM, SPICE, BOOTSTRAP, etc.
  - Bottom-up approaches
    - Which mainly apply measurement as their basic guide for improvement
    - Example: GQM
- Very useful to combine two approaches
  - For example, GQM with CMM
  - GQM gives the answer why we measure an attribute
  - CMM tells us if we are capable of measuring it in a meaningful way

GQM Measurement Phases

- Planning
  - The project for measurement application is selected, defined, characterized, and planned, resulting a project plan
- Definition
  - The measurement program is defined (goal, questions, metrics and hypotheses are defined) and documented
- Data Collection
  - The actual data collection takes place, resulting in collected data
- Interpretation
  - The collected data is processed with respect to the defined metrics into measurement results, that provide answers, to the defined questions, after which goal attainment can be evaluated



Attributes of Effective SW Metrics

- *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.
- *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.* 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

Metrics for the Requirements Model

- Function-based metrics: use the function point 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

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 → ☐

Architectural 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 metrics: a function of the number of modules and the number of interfaces between modules

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 [Chi94]:

- weighted methods per class
- depth of the inheritance tree
- number of children
- coupling between object classes
- response for a class
- lack of cohesion in methods

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
- specialization index

Class-Oriented Metrics

The MOOD Metrics Suite [Har98b]:

- Method inheritance factor
- Coupling factor
- Polymorphism factor

Operation-Oriented Metrics

Proposed by Lorenz and Kidd [Lor94]:

- average operation size
- operation complexity
- average number of parameters per operation

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)

Interface Design Metrics

- Layout appropriateness: a function of layout entities, the geographic position and the "cost" of making transitions among entities

Design Metrics for WebApps

- Does the user interface promote usability?
- Are the aesthetics of the WebApp appropriate for the application domain and pleasing to the user?
- Is the content designed in a manner that imparts the most information with the least effort?
- Is navigation efficient and straightforward?
- Has the WebApp architecture been designed to accommodate the special goals and objectives of WebApp users, the structure of content and functionality, and the flow of navigation required to use the system effectively?
- Are components designed in a manner that reduces procedural complexity and enhances the correctness, reliability and performance?

Code Metrics

- Halstead's Software Science: a comprehensive collection of metrics all predicated on the number (count and occurrence) of operators and operands within a component or program
  – It should be noted that Halstead's "laws" have generated substantial controversy, and many believe that the underlying theory has flaws. However, experimental verification for selected programming languages has been performed (e.g. [FEL89]).

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).

Maintenance Metrics

- IEEE Std. 982.1-1988 [IEE94] suggests a *software maturity index* (SMI) that provides an indication of the stability of a software product (based on changes that occur for each release of the product). The following information is determined:
    - M_T = the number of modules in the current release
    - F_c = the number of modules in the current release that have been changed
    - F_a = the number of modules in the current release that have been added
    - F_d = the number of modules from the preceding release that were deleted in the current release
- The software maturity index is computed in the following manner:
    - $SMI = [M_T - (F_a + F_c + F_d)]/M_T$
- As SMI approaches 1.0, the product begins to stabilize.

Q & A