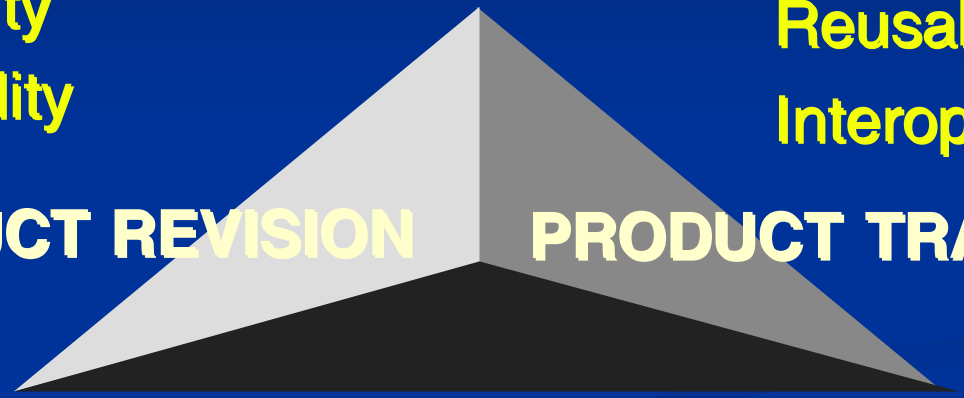
Software Engineering: A Practitioner's Approach, 6/e

Chapter 15
Product Metrics for Software

McCall's Triangle of Quality, 1977(!)

Maintainability
Flexibility
Testability

Portability
Reusability
Interoperability



PRODUCT REVISION

PRODUCT TRANSITION

PRODUCT OPERATION

Correctness
Reliability
Usability
Integrity
Efficiency

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
- 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
  - 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

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}.

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.

programming language independent.

- Metrics should be based on the analysis model, the design model, or the structure of the program itself.

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

Analysis Metrics

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

Function-Based Metrics

- The *function point metric* (FP), first proposed by Albrecht, 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	→					

Design Metrics

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

Class-Oriented Metrics - 1

CK-Metrics: Proposed by Chidamber and Kemerer:

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

Class-Oriented Metrics - 2

LK-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

Operation-Oriented Metrics

Proposed by Lorenz and Kidd [LOR94]:

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

Class-Oriented Metrics - 3

The MOOD Metrics Suite

- Method inheritance factor
- Coupling factor
- Polymorphism factor

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

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).