Chapter 30

- **Product Metrics**

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 8/e
by Roger S. Pressman

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 (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}.

Goal-Oriented Software Measurement

Analyze the *SafeHome* software architecture **for the purpose of** evaluating architectural components **with respect to** the ability to make *SafeHome* more extensible **from the viewpoint of** the software engineers performing the work **in the context of** product enhancement over the next three years.

Q_1: Are architectural components characterized in a manner that compartmentalizes function and related data?

Q_2: Is the complexity of each component within bounds that will facilitate modification and extension?

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.
- *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							

$$\text{FP} = \text{count total} \times [0.65 + 0.01 \times \Sigma\, (F_i)]$$

Function Points

1. Does the system require reliable backup and recovery?
2. Are specialized data communications required to transfer information to or from the application?
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 online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?
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?

Architectural Design Metrics

- Architectural design metrics
  - Structural complexity = g(fan-out)

$$S(i) = f^2_{\text{out}}(i)$$

  - Data complexity = f(input & output variables, fan-out)

$$D(i) = \frac{v(i)}{f_{\text{out}}(i) + 1}$$

  - System complexity = h(structural & data complexity)

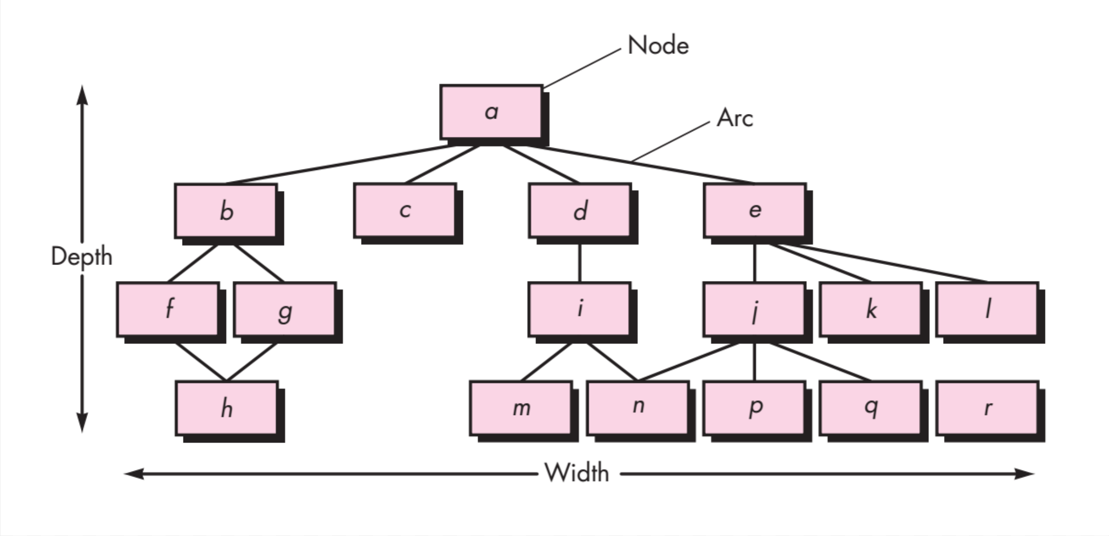
$$C(i) = S(i) + D(i)$$

Architectural Design Metrics

- Morphology metrics: a function of the number of modules and the number of interfaces between modules

$$\text{Size} = n + a$$

$$\text{arc-to-node ratio, } r = a/n,$$



Architectural Design Metrics

- Design Structure Quality Index (DSQI)

S_1 = total number of modules defined in the program architecture

S_2 = number of modules whose correct function depends on the source of data input or that produce data to be used elsewhere (in general, control modules, among others, would not be counted as part of S_2)

S_3 = number of modules whose correct function depends on prior processing

S_4 = number of database items (includes data objects and all attributes that define objects)

S_5 = total number of unique database items

S_6 = number of database segments (different records or individual objects)

S_7 = number of modules with a single entry and exit (exception processing is not considered to be a multiple exit)

Architectural Design Metrics

- Design Structure Quality Index

Program structure: D_1, where D_1 is defined as follows: If the architectural design was developed using a distinct method (e.g., data flow-oriented design or object-oriented design), then $D_1 = 1$, otherwise $D_1 = 0$.

Module independence: $D_2 = 1 - \frac{S_2}{S_1}$

Modules not dependent on prior processing: $D_3 = 1 - \frac{S_3}{S_1}$

$$\boxed{\text{DSQI} = \Sigma\, w_i D_i}$$

Database size: $D_4 = 1 - \frac{S_5}{S_4}$

Database compartmentalization: $D_5 = 1 - \frac{S_6}{S_4}$

Module entrance/exit characteristic: $D_6 = 1 - \frac{S_7}{S_1}$

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

weighted methods per class

Metrics	measurement	توضیح
WMC	$WMC = \sum c_j$	c_j میزان پیچیدگی هر متد (در مجموع برای کل متدها متریک حساب می‌شود) WMC باید تا حد ممکن پایین نگه داشته شود، زیرا: • پیچیده تر شدن درخت وراثت در کلاس‌ها (تمام زیر کلاس ها متدهای کلاس پدر خود را به ارث می برند.) • specific شدن کلاس و کاهش reusability

Class-Oriented Metrics

- **Depth of the inheritance tree (DIT)**

Metrics	measurement	توضیح
DIT	طولانی ترین مسیر از یک node تا ریشه	با افزایش آن ، کلاس‌های سطح پایین خیلی تخصصی می شوند (امکان reuse کم) باید متدهای زیادی را به ارث ببرند، پیچیدگی زیاد در صورت اضافه یک متد به parent لازم است برای تعداد زیادی از کلاس‌ها بررسی شود که آیا متد اضافه شده باید override شود یا خیر.

Class-Oriented Metrics

- **Number of children (NOC)**

Metrics	measurement	توضیح
NOC	تعداد فرزندان immediate یک کلاس	هرچه **NOC** بیشتر شود effort لازم برای تست بیشتر می‌شود.

Class-Oriented Metrics

- **Coupling between object classes (CBO)**

Metrics	measurement	توضیح
CBO	تعداد collaborator های موجود برای یک کلاس	• هر چه CBO بیشتر شود میزان reusability و modifiability و testability کم. (cost of change بیشتر میشود) • باید تا حد ممکن کم نگه داشته شود.

Class-Oriented Metrics

- **Response for a class (RFC)**

Metrics	measurement	توضیح
RFC	تعداد متدهایی که در پاسخ به یک message دریافت شده در یک object فراخوانی می‌شوند.	هرچه **RFC** بیشتر شود effort لازم برای تست بیشتر می‌شود. (test sequence بیشتر) پیچیدگی طراحی بیشتر

Class-Oriented Metrics

- **Lack of cohesion in methods (LCOM)**

Metrics	measurement	توضیح
LCOM	تعداد متدهایی که به یک یا چند attribute یکسان دسترسی دارند.	با افزایش LCOM میزان coupling متدها از طریق attribute ها افزایش می‌یابد. (cohesion در سطح متد کاهش می‌یابد.) بنابراین LCOM حداقل مورد انتظار است.

Class-Oriented Metrics

The MOOD Metrics Suite [Har98b]:

Metrics	measurement	توضیح
MIF	$$\frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$ میزان درجه ای که ساختار کلاس در سیستم OO از وراثت استفاده می‌کند (هم در رابطه با صفات و هم در رابطه با متدها)	• TC= تعداد کل کلاس • هدف از این معیار • برای هر کلاس C_i ○ $M_d(C_i)$: تعداد متدهایی کلاس C_i ○ $M_i(C_i)$: تعداد متدهای که به ارث برده شده اما override نشده ○ $M_a(C_i)$: تعداد متدهای که از طریق سایر کلاس در رابطه association قابل فراخوانی هستند.

Class-Oriented Metrics

The MOOD Metrics Suite [Har98b]:

- **Coupling factor (CF).**

$$\mathrm{CF} = \Sigma_i\, \Sigma_j\, is\_client\, \frac{(C_i, C_j)}{T_c^{\,2} - T_c}$$

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)

Component-Level Design Metrics

metric	Sub-metric	measurement	توضيح
module coupling (m_c)	data and control flow coupling	d_i= number of input data parameter	
		c_i= number of input control parameters	
		d_o = number of output data parameters	
		c_o = number of output control parameters	
	global coupling	g_d = number of global variables used as data	
		g_c = number of global variables used as control	
	environmental coupling	W= number of modules called (fan-out) r = number of modules calling the module under consideration (fan-in)	
	$m_c = \dfrac{k}{M}$ where k is a proportionality constant and $M = d_i + (a \times c_i) + d_o + (b \times c_o) + g_d + (c \times g_c) + w + r$		

Operation-Oriented Metrics

Proposed by Lorenz and Kidd [Lor94]:

- **Average operation size (OS avg).**
  - # lines of code
  - number of messages sent by the operation.
- **Operation complexity (OC).**
- **Average number of parameters per operation (NP avg).**

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?

Design Metrics for WebApps

- Interface Metrics

Suggested Metric	*Description*
Layout appropriateness	
Layout complexity	Number of distinct regions[12] defined for an interface
Layout region complexity	Average number of distinct links per region
Recognition complexity	Average number of distinct items the user must look at before making a navigation or data input decision
Recognition time	Average time (in seconds) that it takes a user to select the appropriate action for a given task
Typing effort	Average number of key strokes required for a specific function
Mouse pick effort	Average number of mouse picks per function
Selection complexity	Average number of links that can be selected per page
Content acquisition time	Average number of words of text per Web page
Memory load	Average number of distinct data items that the user must remember to achieve a specific objective

Design Metrics for WebApps

- **Aesthetic (graphic design) metrics**

Suggested Metric	Description
Word count	Total number of words that appear on a page
Body text percentage	Percentage of words that are body versus display text (i.e., headers)
Emphasized body text %	Portion of body text that is emphasized (e.g., bold, capitalized)
Text positioning count	Changes in text position from flush left
Text cluster count	Text areas highlighted with color, bordered regions, rules, or lists
Link count	Total links on a page
Page size	Total bytes for the page as well as elements, graphics, and style sheets
Graphic percentage	Percentage of page bytes that are for graphics
Graphics count	Total graphics on a page (not including graphics specified in scripts, applets, and objects)
Color count	Total colors employed
Font count	Total fonts employed (i.e., face + size + bold + italic)

Design Metrics for WebApps

- Content Metrics

Suggested Metric	Description
Page wait	Average time required for a page to download at different connection speeds
Page complexity	Average number of different types of media used on page, not including text
Graphic complexity	Average number of graphics media per page
Audio complexity	Average number of audio media per page
Video complexity	Average number of video media per page
Animation complexity	Average number of animations per page
Scanned image complexity	Average number of scanned images per page

Design Metrics for WebApps

- Navigation Metrics

Suggested Metric	Description
Page-linking complexity	Number of links per page
Connectivity	Total number of internal links, not including dynamically generated links
Connectivity density	Connectivity divided by page count

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]).

Code Metrics

- Halstead's Metrics

Metric	measurement	Primitive measurements
overall program length	N = n1 log2 n1 + n2 log2 n2	n1 = number of distinct operators that appear in a program n2= number of distinct operands that appear in a program N1 = total number of operator occurrences N2 = total number of operand occurrences
program volume • حجم اطلاعات(بر منبای bit) مورد نیاز برای specify کردن یک برنامه	V = N log2 (n1+n2)	

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.