Chapter 30

- **Product Metrics**

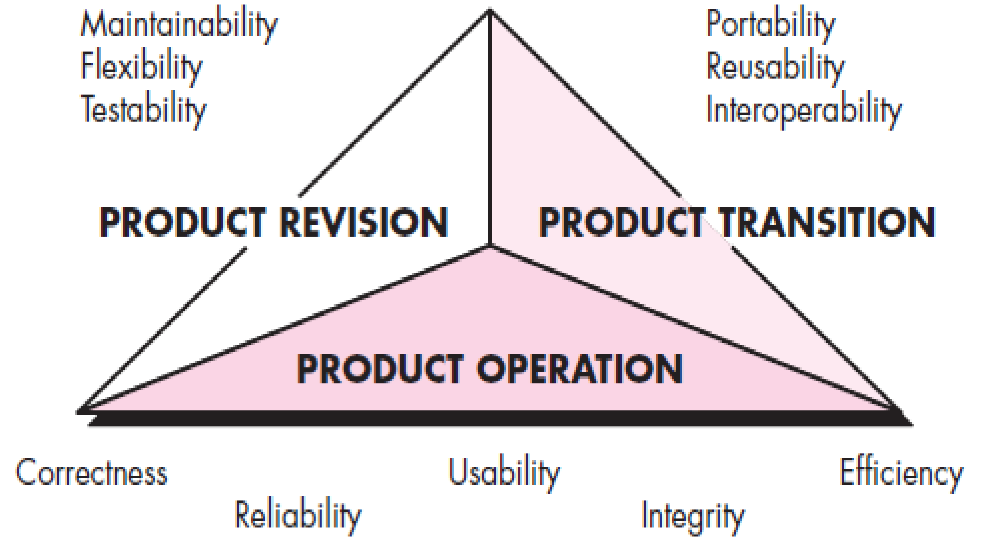
Slide Set to accompany
Software Engineering: A Practitioner’s Approach
by Roger S. Pressman

Table of Contents

30.1 A Framework for Product Metrics

FIGURE 14.1

McCall's software quality factors



“They are as valid today as they were in that time.….” - Pressman

☛ ***Further, system quality and development quality need be “statistically” or “quantitatively managed.***

30.1.1 Measures, Metrics and Indicators

- Measure
  - Provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process

  E.g. Kg, cm, KLOC, ...

- Metric
  - “A quantitative measure of the degree to which a system, component, or process possesses a given attribute.”

    - [IEEE 93] IEEE Standard Glossary of Software Engineering Terminology

- Indicator
  - A metric or combination of metrics that provide insight into the software process, a software project, or the product itself

30.1.2 The Challenges of Product Metrics

- In the past, researchers attempted to develop a single metric that provides a comprehensive measure of software complexity.
- ☛ [Fen 94] characterizes this research as a search for "the impossible holy grail."
- Is it possible to develop measures of different internal program attributes (e.g. effective modularity, functional independence, etc.)?
- ☛ "The danger of attempting to find measures which characterize so many different attributes is that inevitably the measure have to satisfy conflicting aims." - [Fen 94]
- However, see the next slide.

Example: Measuring a business Value

Value: Timeliness of change request processing from the project manager's viewpoint
=> variables: CRT

An indicator

A metric

CRT := PMSatisfaction / (ACT × SD × FaultRate) ------ (C1)

- KPI: Current change request processing speed
  - KPI: Average cycle time => variables: ACT
  - KPI: Standard deviation => variables: SD
  - KPI: % cases outside of the upper limit => variables: FaultRate
- Subvalue: The project manager is satisfied with the current request processing speed
  - Subjective judgment: subjective evaluation by Project Manager
    => variables: PMSatisfaction

Need metrics

KPI: Key Performance Indicator

Source: Figure 8 in
Sungwon Kang, Jihyun Lee, Dan Hyung Lee, Jongmoon Baik, "A Framework for Measuring and Managing Value Achievement in Business Processes," *IEICE Transactions on Information and Software Systems*. Vol.E95-D, No.10, pp.2456-2468, Oct. 2012.

30.1.3 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 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 [Basili 84]

Measurement Process

1. Formulation

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

2. Collection

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

3. Analysis

   The computation of metrics and the application of mathematical tools.

4. Interpretation

   The evaluation of metrics results in an effort to gain insight into the quality of the representation.

5. Feedback

   Recommendations derived from the interpretation of product metrics transmitted to the software team.

30.1.4 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}.

30.1.5 The Attributes of Effective Software 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

30.2 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

30.2.1 Function-Based Metrics

- The Function Point (FP) metric
  - Proposed by [Albrecht 79]
  - An effective means for measuring the functionality delivered by a system.
  - Derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity
- Information domain values consists of the counts of :
  - External inputs (EIs)
  - External outputs (EOs)
  - External inquiries (EQs)
  - Internal logical files (ILFs)
  - External interface files (EIFs)

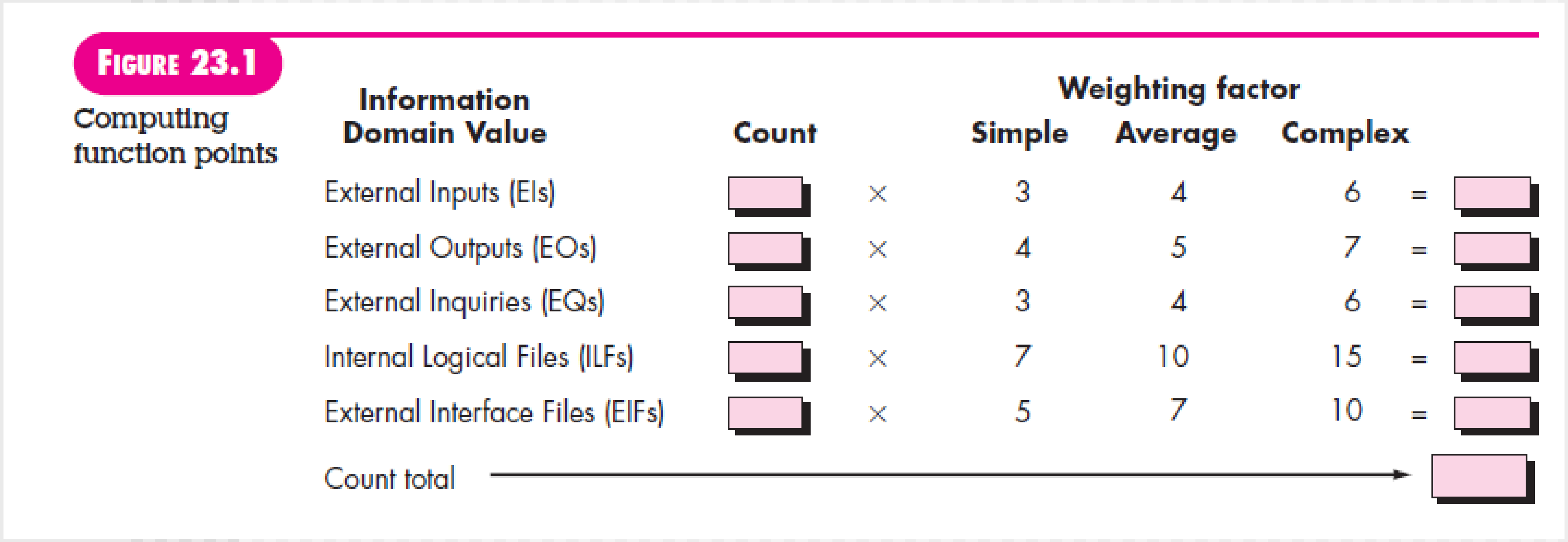


FIGURE 23.1 Computing function points

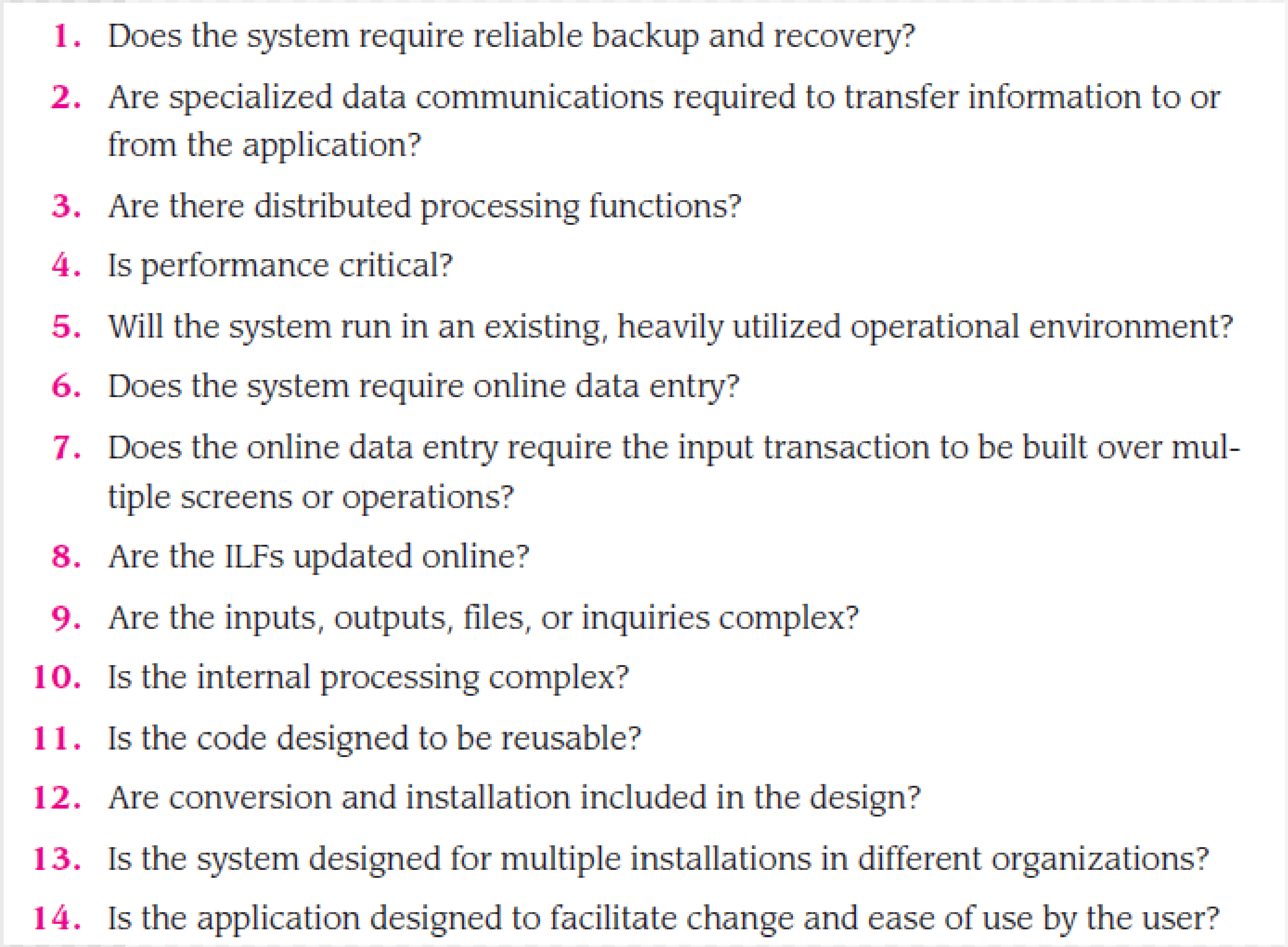
Information Domain Value	Count		Weighting factor Simple	Average	Complex		
External Inputs (EIs)		×	3	4	6	=	
External Outputs (EOs)		×	4	5	7	=	
External Inquiries (EQs)		×	3	4	6	=	
Internal Logical Files (ILFs)		×	7	10	15	=	
External Interface Files (EIFs)		×	5	7	10	=	
Count total						→	

$$\underbrace{\text{FP}}_{\text{Adjusted FP}} = \underbrace{\text{count total}}_{\text{Unadjusted FP}} \times [0.65 + 0.01 \times \underbrace{\Sigma (F_i)]}_{\text{value adjustment factors (VAF)}}$$

Each rated on scales equivalent to the following:

Not present	= 0
Incidental Influence	= 1
Moderate Influence	= 2
Average Influence	= 3
Significant Influence	= 4
Strong Influence	= 5

value adjustment factors (VAF)



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?

FIGURE 23.2

A data flow model for *SafeHome* software

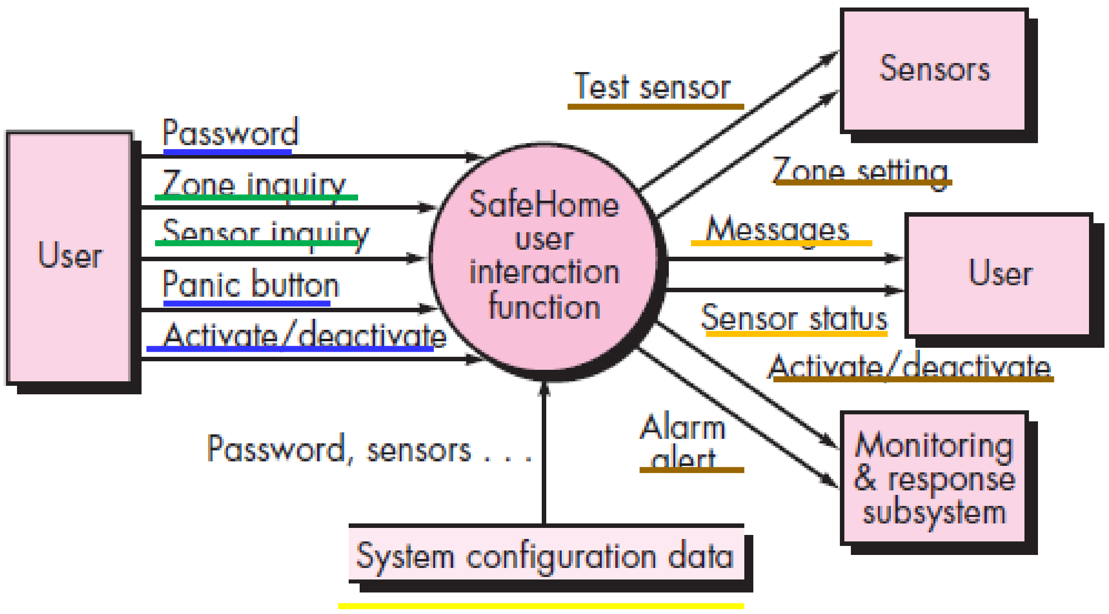
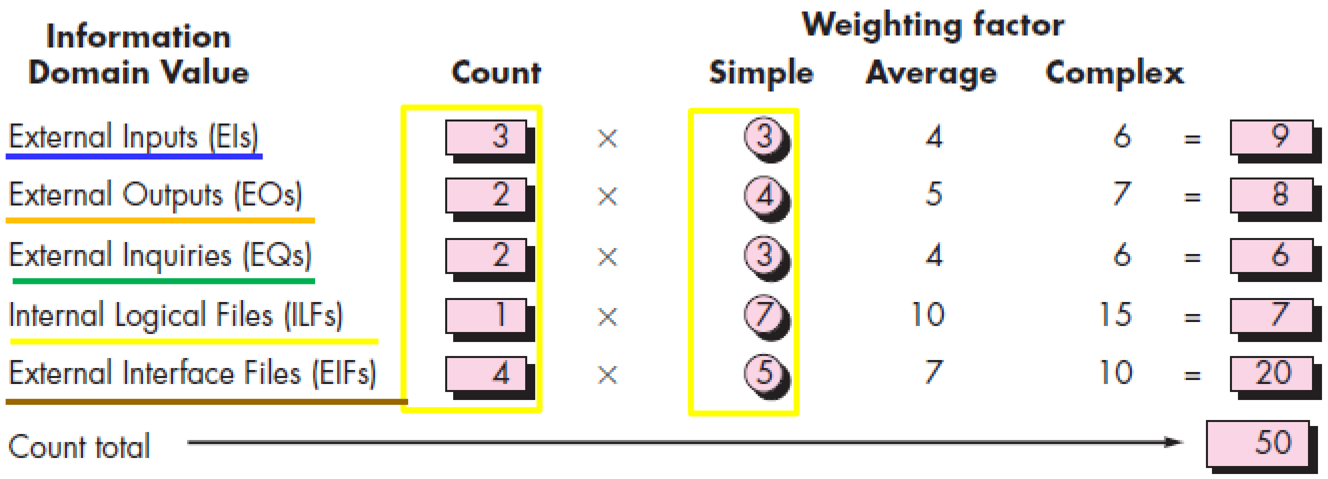


FIGURE 23.3

Computing function points



Information Domain Value	Count		Weighting factor Simple	Average	Complex		
External Inputs (EIs)	3	×	3	4	6	=	9
External Outputs (EOs)	2	×	4	5	7	=	8
External Inquiries (EQs)	2	×	3	4	6	=	6
Internal Logical Files (ILFs)	1	×	7	10	15	=	7
External Interface Files (EIFs)	4	×	5	7	10	=	20
Count total							50

Assume $\Sigma(F_i)$ is 46

Then $FP = 50 \times [0.65 + (0.01 \times 46)] = 56$

- FPs can be used to estimate LOC depending on the AVerage number of LOC (AVC) per FP for a given language
  - LOC = AVC * number of function points
  - AVC: 200 ~ 300 for assemble language
    2 ~ 40 for a 4GL

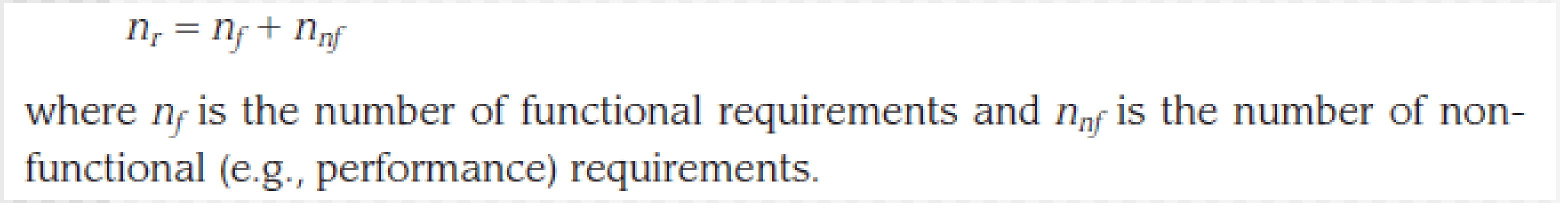
How many debugged LOC can you write per day?

30.2.2 Metrics for Specification Quality

- Specificity (lack of ambiguity)
- Completeness
- Correctness
- Understandability
- Verifiability
- Internal and external consistency
- Achievability
- Concision
- Traceability
- Modifiability
- Precision
- Reusability

[Dav 93]

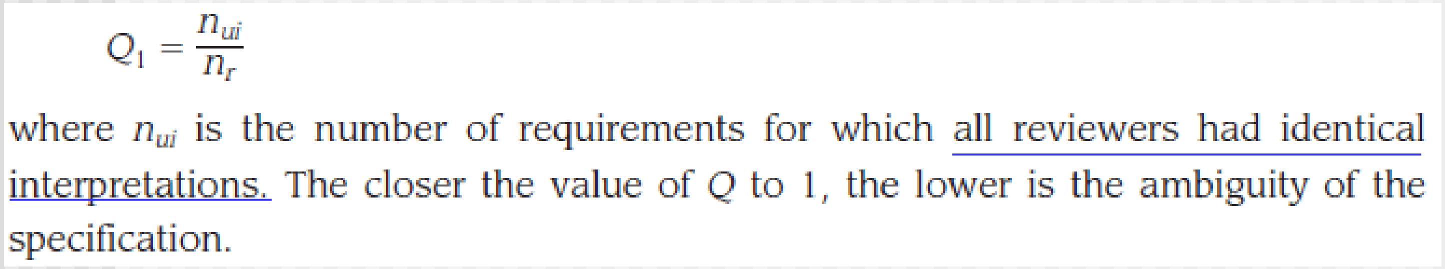
Assume



$$n_r = n_f + n_{nf}$$

where n_f is the number of functional requirements and n_{nf} is the number of non-functional (e.g., performance) requirements.

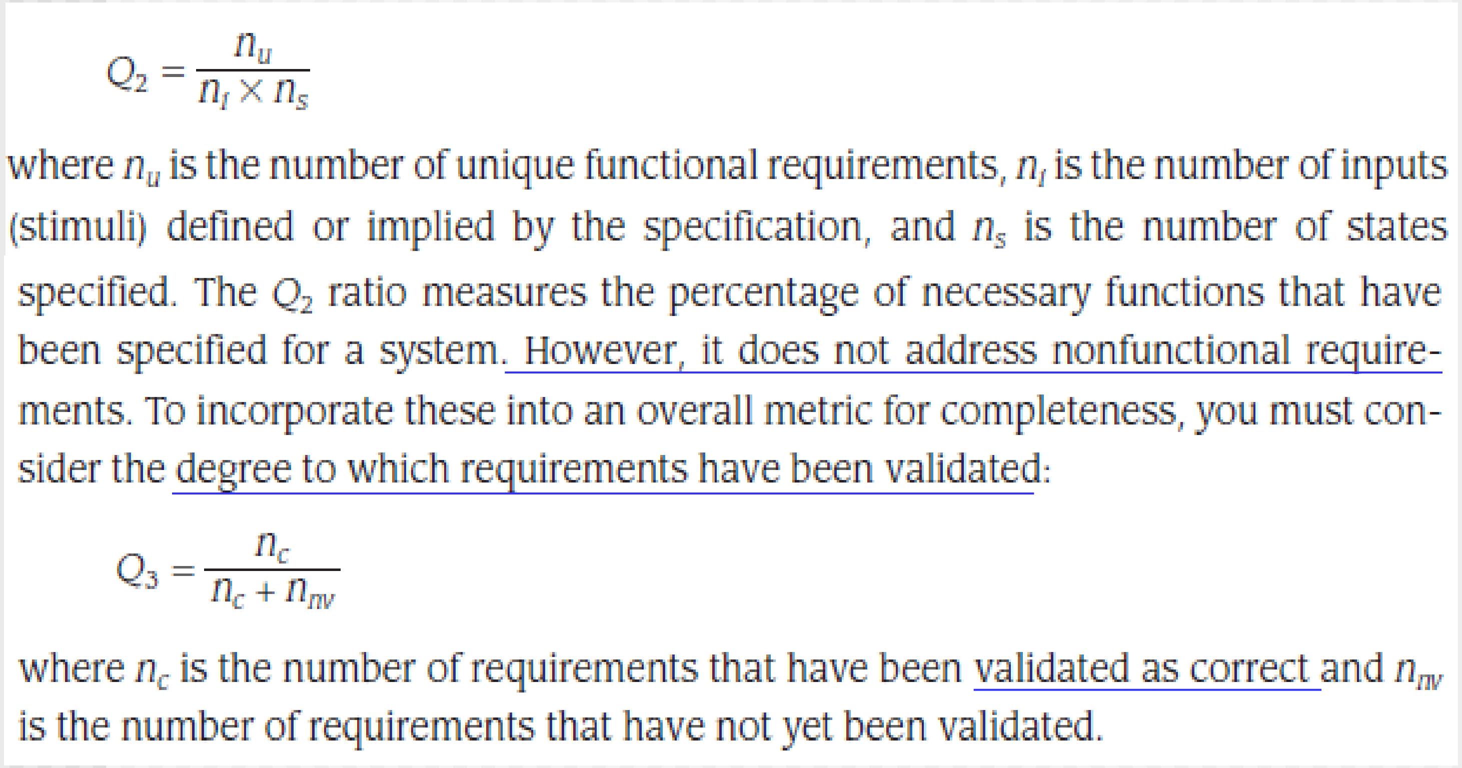
Then specificity can be defined as



$$Q_1 = \frac{n_{ui}}{n_r}$$

where n_{ui} is the number of requirements for which all reviewers had identical interpretations. The closer the value of Q to 1, the lower is the ambiguity of the specification.

Completeness of functional requirements can be defined as



$$Q_2 = \frac{n_u}{n_I \times n_s}$$

where n_u is the number of unique functional requirements, n_I is the number of inputs (stimuli) defined or implied by the specification, and n_s is the number of states specified. The Q_2 ratio measures the percentage of necessary functions that have been specified for a system. However, it does not address nonfunctional requirements. To incorporate these into an overall metric for completeness, you must consider the degree to which requirements have been validated:

$$Q_3 = \frac{n_c}{n_c + n_{nv}}$$

where n_c is the number of requirements that have been validated as correct and n_{nv} is the number of requirements that have not yet been validated.

30.3 Metrics for the Design Model

30.3.1 Architectural Design Metrics

- Architectural design metrics(Card and Glass [Card 90])
  - Structural complexity

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

where $f_{\text{out}}(i)$ is the fan-out of module i.

fan-out(i): number of modules that are directly invoked by module i

  - Data complexity

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

where $v(i)$ is the number of input and output variables that are passed to and from module i.

  - System complexity

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

- HK(Henry-Kafura) Metric: architectural complexity as a function of fan-in and fan-out

HKM = *length*(i) x $[f_{in}(i) + f_{out}(i)]^2$

length(i) = The number of statements in module i

f_{in} = fan-in for module i

f_{out} = fan-out for module i

fan-in(i): number of modules that directly invoke module i

- Morphology metrics([Fenton 91]): a function of the number of modules and the number of interfaces between modules

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

where n is the number of nodes and a is the number of arcs.

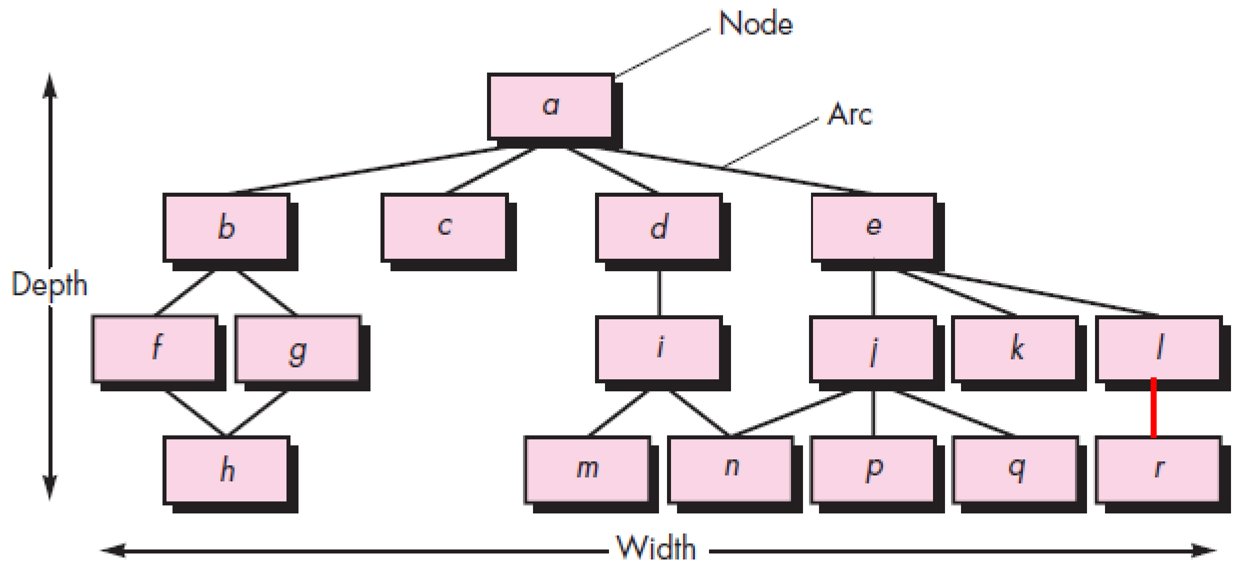
Depth = longest path from the root (top) node to a leaf node.

Width = maximum number of nodes at any one level of the architecture.

arc-to-node ratio, $r = a/n$, measures the connectivity density of the architecture

FIGURE 23.4

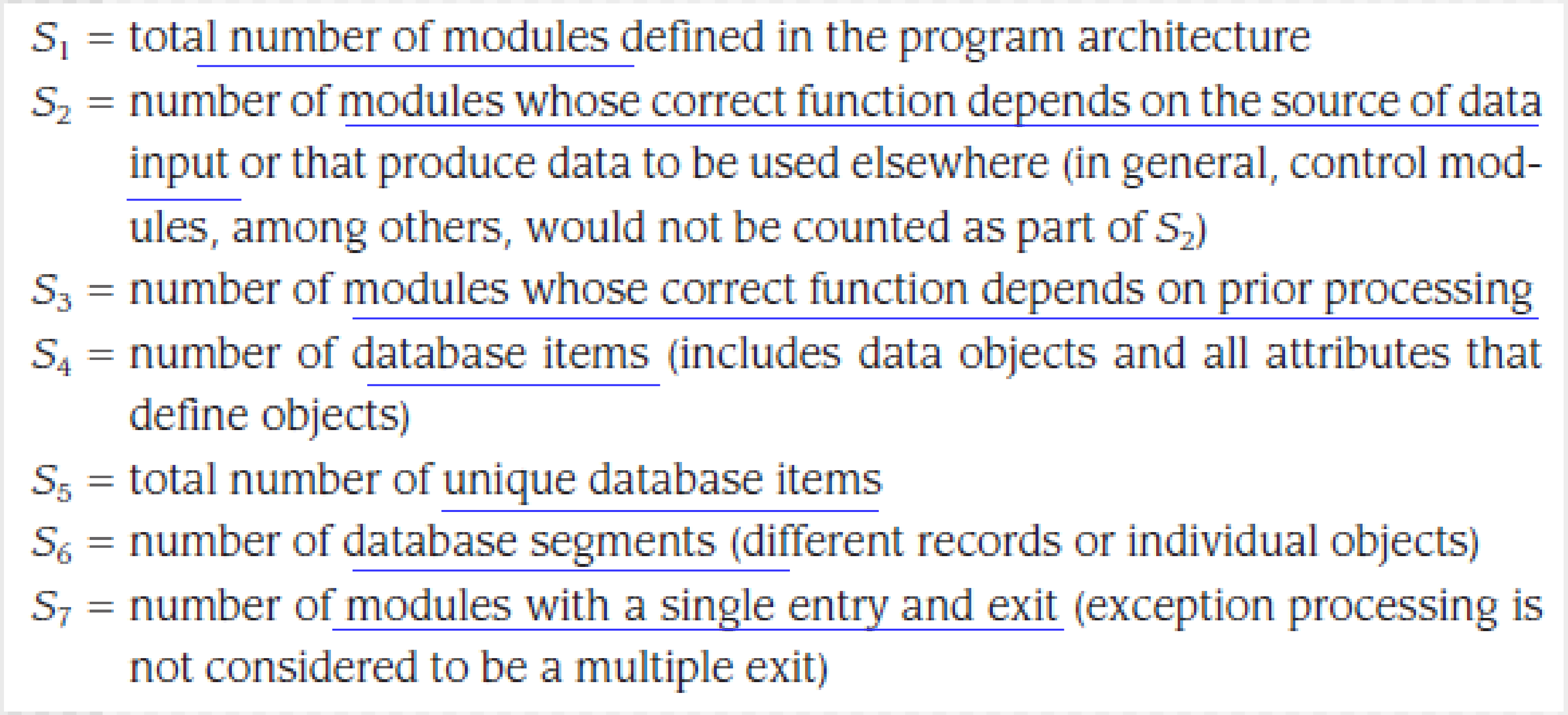
Morphology metrics



Size	=	#nodes + #Arcs = 17 + 18 = 35
Depth	=	Longest path from the root to a leaf = 4
Width	=	Maximum number of nodes at one level = 6
R	=	#arcs / #nodes = 18/17 = 1.06

US Air Force Systems Command[USA 87]

- Design Structure Quality Indicator (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)

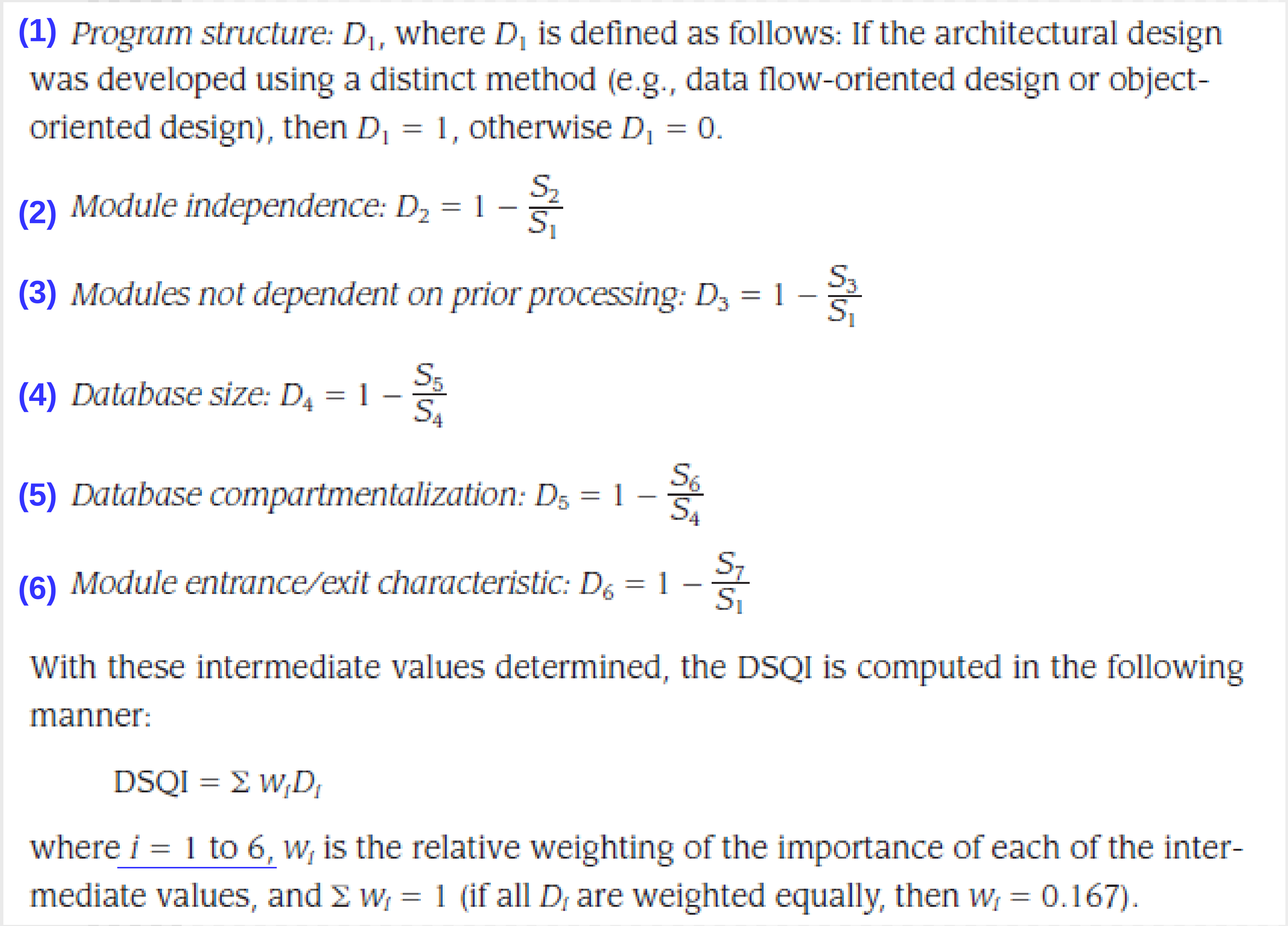
S_3 = number of modules whose correct function depends on prior processing

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)



(1) *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 $D_1 = 0$.

(2) *Module independence:* $D_2 = 1 - \frac{S_2}{S_1}$

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

(4) *Database size:* $D_4 = 1 - \frac{S_5}{S_4}$

(5) *Database compartmentalization:* $D_5 = 1 - \frac{S_6}{S_4}$

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

With these intermediate values determined, the DSQI is computed in the following manner:

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

where $i = 1$ to 6, w_i is the relative weighting of the importance of each of the intermediate values, and $\Sigma\, w_i = 1$ (if all D_i are weighted equally, then $w_i = 0.167$).

30.3.2 Metrics for OO Design

- [Whitmire 97] describes 9 distinct and measurable characteristics of an OO design:
  1. Size

     Size is defined in terms of four views: population, volume, length, and functionality
  2. Complexity

     How classes of an OO design are interrelated to one another
  3. Coupling

     The physical connections between elements of the OO design
  4. 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."
  5. Completeness

     An indirect implication about the degree to which the abstraction or design component can be reused

6. Cohesion

   The degree to which all operations working together to achieve a single, well-defined purpose

7. Primitiveness

   Applied to both operations and classes, the degree to which an operation is atomic

8. Similarity

   The degree to which two or more classes are similar in terms of their structure, function, behavior, or purpose

9. Volatility

   Measures the likelihood that a change will occur

[Berard 95] 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

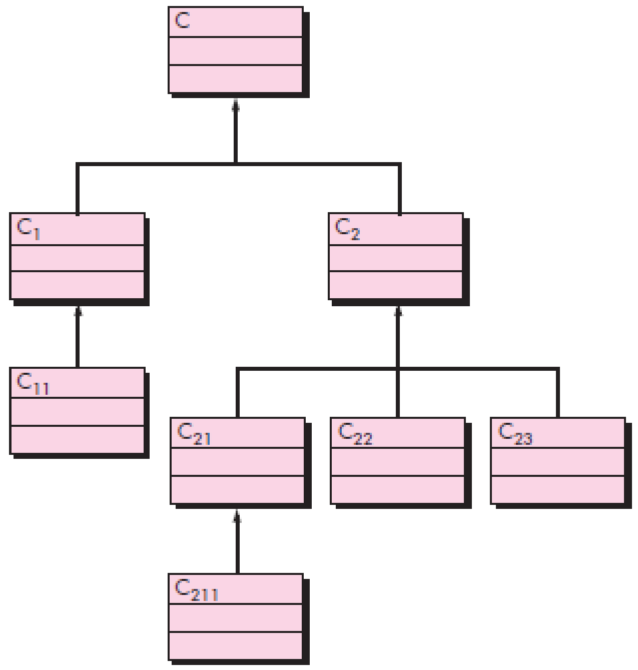
30.3.3 Class-Oriented Metrics – The CK Metrics Suite

"The validity of some of the metrics discussed in this chapter is currently debated in the technical literature." - Pressman

- Proposed by Chidamber and Kemerer [Chidamber 94]:

FIGURE 23.5

A class hierarchy



6 metrics of The CK Metrics Suite

1. Weighted methods per class: The number of methods and their complexity are reasonable indicators of the amount of effort required to implement and test a class
2. Depth of the inheritance tree: maximum length form the node to the root of the tree

   E.g. DIT = 4
3. Number of children:

   E.g. C2 has 3 children: C21, C22, C23
4. Coupling between object classes: the number of collaboration listed for a class on its CRC card
5. Response for a class: the set of methods that can potentially be executed in response to a message received by an object of that class
6. Lack of cohesion in methods: the number of methods that access one or more of the same attributes.

30.3.4 Class-Oriented Metrics - *The MOOD Metrics Suite*

[Harmon 98b]:

- Method inheritance factor (MIF)
- Coupling factor (CF)
- Polymorphism factor

30.3.5 OO Metrics *Proposed by Lorenz and Kidd*

[Lorenz 94]:

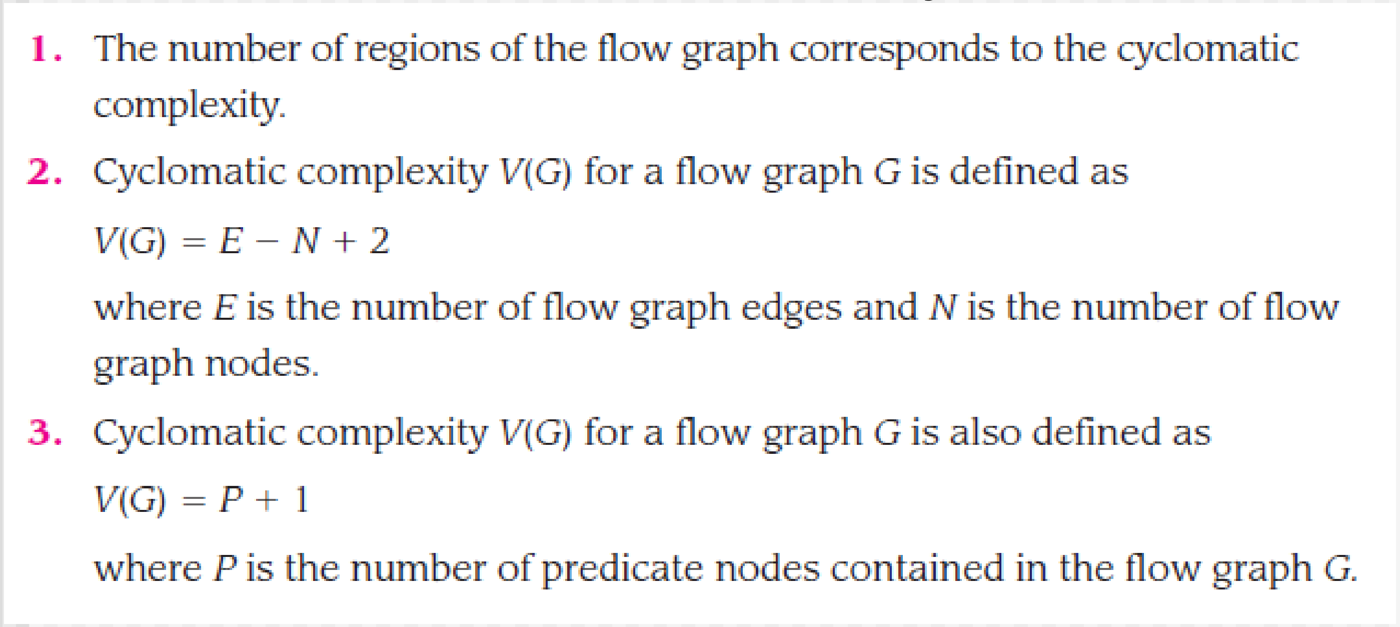
- Class size
- Number of operations overridden by a subclass
- Number of operations added by a subclass
- Specialization index

30.3.6 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**)

Cyclomatic Complexity (= CC)

- Provides a quantitative measure of the logical complexity of a program
- Measured in one of the three ways



1. The number of regions of the flow graph corresponds to the cyclomatic complexity.
2. Cyclomatic complexity $V(G)$ for a flow graph G is defined as

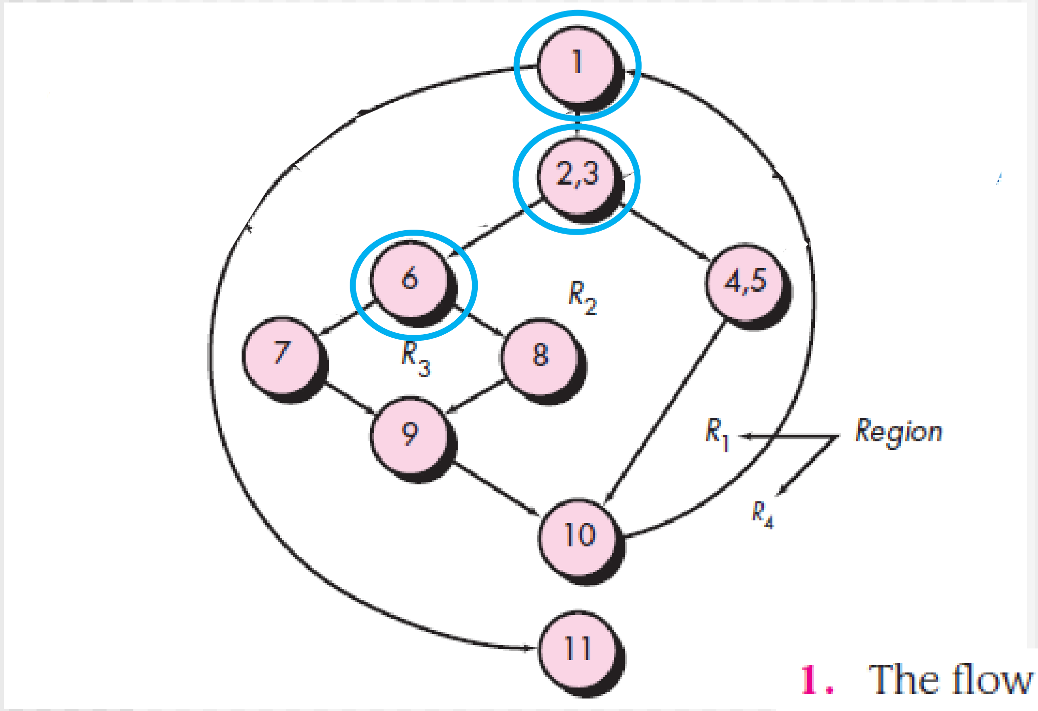
   $V(G) = E - N + 2$

   where E is the number of flow graph edges and N is the number of flow graph nodes.
3. Cyclomatic complexity $V(G)$ for a flow graph G is also defined as

   $V(G) = P + 1$

   where P is the number of predicate nodes contained in the flow graph G.

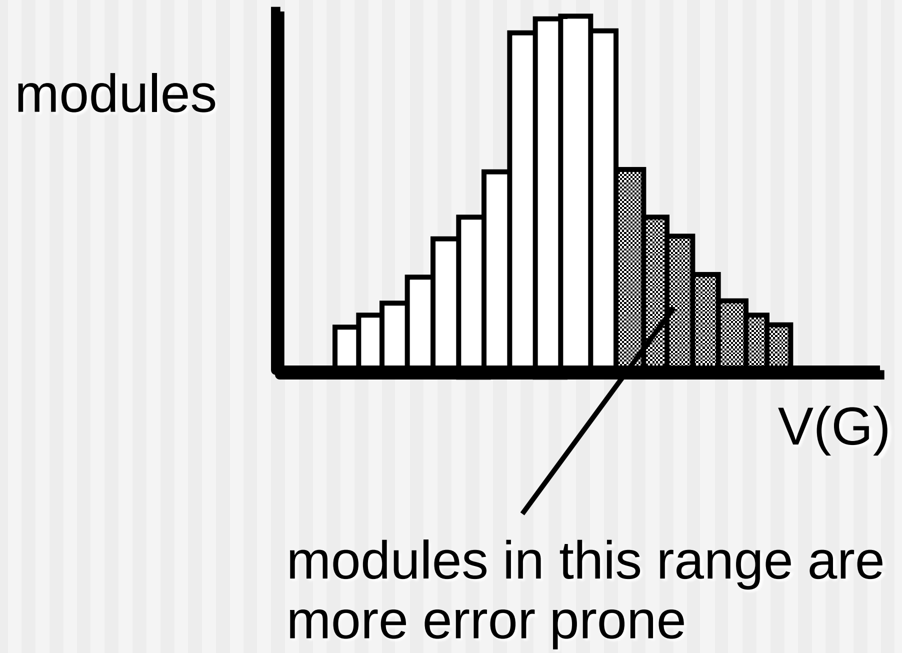
Cyclomatic Complexity



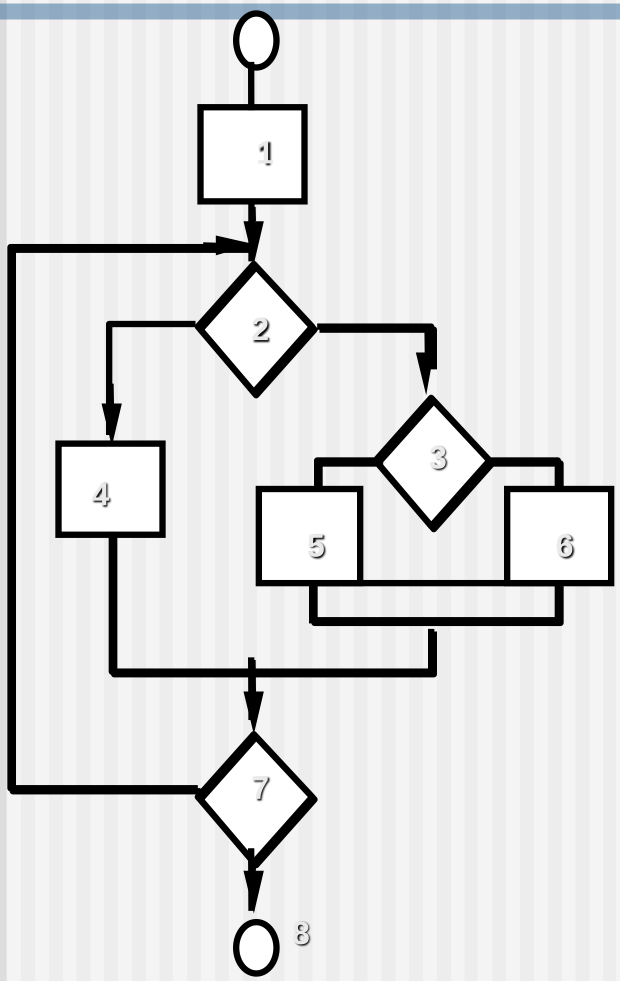
1. The flow graph has four regions.
2. $V(G) = 11$ edges $- 9$ nodes $+ 2 = 4$.
3. $V(G) = 3$ predicate nodes $+ 1 = 4$.

Cyclomatic Complexity

A number of industry studies have indicated that the higher V(G), the higher the probability or errors.



Basis Path Testing



V(G)
= Number of simple decisions + 1
= Number of enclosed areas + 1
= 4

Next, we derive the independent paths:

Since V(G) = 4,
there are four paths

Path 1:	1,2,3,6,7,8
Path 2:	1,2,3,5,7,8
Path 3:	1,2,4,7,8
Path 4:	1,2,4,7,2,4,...7,8

Finally, we derive test cases to exercise these paths.

30.3.7 Operation-Oriented Metrics

- Proposed by Lorenz and Kidd [Lor94]:
  - Average operation size
  - Operation complexity
  - Average number of parameters per operation

30.3.8 User Interface Design Metrics

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

30.5 Metrics for Source Code (1/2)

- Halstead's Software Science: a comprehensive collection of metrics all predicated on
  - The number (count and occurrence) of operators
  - The number of operands

  within a component or program

- 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. [Feller 89]).

30.5 Metrics for Source Code(2/2)

- Testing effort can also be estimated using metrics derived from Halstead measures

n_1 = number of distinct operators that appear in a program
n_2 = number of distinct operands that appear in a program
N_1 = total number of operator occurrences
N_2 = total number of operand occurrences

Program Length: $N = n_1 \log_2 n_1 + n_2 \log_2 n_2$

Program Volume: $V = N \log_2 (n_1 + n_2)$

For a given algorithm, a minimum volume must exist.

Volume Ratio: $L = \frac{2}{n_1} \times \frac{n_2}{N_2}$ < 1

30.6 Metrics for Testing

30.6.1 Halstead Metrics Applied to Testing

Program Level: $$PL = \frac{1}{(n_1/2) \times (N_2/n_2)}$$

Halstead (testing) effort: $$e = \frac{V}{PL}$$

For a module k, $$\text{Percentage of testing effort } (k) = \frac{e(k)}{\Sigma e(i)}$$

30.6.2 Metrics for OO Testing

- [Binder 94] 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)
  - Depth of the inheritance tree (DIT).

30.7 Metrics for Maintenance

- IEEE Std. 982.1-1988 [IEEE 94] suggests a software maturity index (SMI) that indicates the stability of a software product.

  Let

  M_T = # modules in the current release
  F_c = # modules in the current release that have been changed
  F_a = # modules in the current release that have been added
  F_d = # modules from the preceding release that were deleted

- Then

  $SMI = [M_T - (F_a + F_c + F_d)]/M_T$

- As SMI approaches 1.0, the product begins to stabilize.