CHAPTER 12: PRODUCT METRICS

SOFTWARE ENGINEERING (UNDERGRADUATE)

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 (e.g. size)
- Function points are derived using an empirical relationship based on countable 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): input transactions that update internal computer files
  - Number of external outputs (EOs): transactions where data is output to the user, e.g. printed reports
  - Number of internal logical files (ILFs): group of data that is usually accessed together, e.g. purchase order file
  - Number of external interface files (EIFs): file sharing among different applications to achieve a common goal
  - Number of external inquiries (EQs): transactions that provide information but do not update internal file

FUNCTION POINTS METRICS

Information Domain Value	**Count**		**Simple**	**Average**	**Complex**		
(FPunadjusted)							
Number of external inputs (EIs)		X	3	4	6	=	
Number of external outputs (EOs)		X	4	5	7	=	
Number of external inquiries (EQs)		X	3	4	6	=	
Number of internal logical files (ILFs)		X	7	10	15	=	
Number of external interface files (EIFs)		X	5	7	10	=	
					Count Total		

REQUIREMENTS QUALITY METRICS

❑ Requirements related to "Reliability" can use different measures to quantify the goal

Property	**Measure**
Reliability	1. Mean (interval) time to failure 2. Rate of failure occurrence

❑ Requirements related to "Robustness" can use different measures to quantify the goal

Property	**Measure**
Robustness	1. Time to restart after failure 2. Percentage of events causing failures 3. Probability of data corruption on failure

METRICS FOR OO DESIGN

Whitmire describes nine distinct and measurable characteristics of an OO design:

- **Size:** size is defined in terms of 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
- **Cohesion :** the degree to which all operations working together to achieve a single, well-defined purpose

METRICS FOR 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. (e.g. deals with interface and hide internals to the users)
- **Completeness:** an indirect implication about the degree to which the abstraction or design component can be reused
- **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

CLASS ORIENTED METRICS

Proposed by Chidamber and Kemerer:

- Weighted methods per class - number of functions in class (WMC)
- Depth of the inheritance tree (DIT)
- Number of children (NOC)
- Coupling between object classes (CBC)
- Lack of cohesion in methods (LCOM)

CLASS ORIENTED METRICS

Proposed by Lorenz and Kidd:

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

OPERATION-ORIENTED METRICS

Proposed by Lorenz and Kidd:

- Average operation size
- Operation complexity
- Average number of parameters per operation

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

METRICS FOR TESTING

- Testing effort can also be estimated using metrics
- Binder 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).
  - Number of children (NOC) and depth of the inheritance tree (DIT).

MAINTENANCE METRICS

- IEEE Std. 982.1-1988 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 = $[MT - (F_a + F_c + F_d)]/M_T$

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

REFERENCES

- R.S. Pressman & Associates, Inc (2010). *Software Engineering: A Practitioner's Approach.*