Note Based on "Chapter 6 - Software Maintenance Metrics and Cost"

Definitions

- **Measure**: A quantitative indication of the extent, amount, dimension, or size of some attribute of a product or process (e.g., number of errors).
- **Metrics**: The degree to which a system, component, or process possesses a given attribute, relating several measures (e.g., average number of errors found per person hour).
- **Indicators**: A metric or combination of metrics that provide insight into the software process, a software project, or the product itself.
- **Direct Metrics**: Immediately measurable attributes (e.g., lines of code, execution speed, defects reported).
- **Indirect Metrics**: Aspects not immediately quantifiable (e.g., functionality, reliability).
- **Measurement**: The process by which numbers or symbols are assigned to attributes of entities in the real world according to clearly defined rules.
- **Faults**:
  - **Errors**: Faults found by developers during software development.
  - **Defects**: Faults found by customers after release.

Software Measurement and Metrics

- **Software Measurement**: A quantitative attribute of a software product or the software process.

- **Software Measurement Process**: Defined and governed by ISO Standard.

- **Metrics**: Measurement of the level that any attribute belongs to a system product or process.

- **Functions related to software metrics**:

  - Planning
  - Organizing
  - Controlling
  - Improving

Classification of Software Measurement

- **Direct Measurement**: The product, process, or thing is measured directly using a standard scale.
- **Indirect Measurement**: The quantity or quality to be measured is measured using a related parameter, i.e., by use of reference.

Characteristics of Software Metrics

- **Quantitative**: Metrics must possess quantitative nature, meaning they can be expressed in values.
- **Understandable**: Metric computation should be easily understood, and the method of computing the metric should be clearly defined.
- **Applicability**: Metrics should be applicable in the initial phases of software development.
- **Repeatable**: The metric values should be consistent when measured repeatedly.
- **Economical**: Computation of metrics should be cost-effective.
- **Language Independent**: Metrics should not depend on any programming language.

Classification of Software Metrics

- **Product Metrics**: Used to evaluate the state of the product, trace risks, and uncover prospective problem areas.
- **Process Metrics**: Focus on enhancing the long-term process of the team or organization.
- **Project Metrics**: Describe project characteristics and execution process, helping to track risk, identify problem areas, and adjust workflow.

Metrics for the Analysis Model

- **Functionality Delivered**: Provides an indirect measure of the functionality packaged within the software.
- **System Size**: Measures the overall size of the system in terms of the information available as part of the analysis model.
- **Specification Quality**: Provides an indication of the specificity and completeness of a requirements specification.

Function Points

- **Function Points**: A means for measuring the functionality delivered by a system.

    - **Computation**:
        a. Identify and collect information domain values.
        b. Complete a table to get the count total.
        c. Evaluate and sum up adjustment factors.
        d. Compute the number of function points (FP): [FP = \text{count total} \times [0.65 + 0.01 \times \text{sum}(Fi)]]

- **Example Calculation**:

    - Given:

    - External Inputs: 3 (Simple)
    - External Outputs: 2 (Average)
    - External Inquiries: 2 (Simple)
    - Internal Logical Files: 1 (Simple)
    - External Interface Files: 4 (Complex)
    - Count total = 50
    - Value adjustment factors sum (Fi) = 46
  - Calculate: [FP = 50 \times [0.65 + (0.01 \times 46)] = 55.5 \ (\text{rounded up to 56})]

- **Interpretation of FP Number**:
  - One FP translates into 60 lines of object-oriented source code.
  - 12 FPs are produced for each person-month of effort.
  - Three errors per function point found during analysis and design reviews.
  - Four errors per function point found during unit and integration testing.

Metrics for the Design Model

- **Architectural Metrics**: Indicate the quality of the architectural design.
- **Component-level Metrics**: Measure the complexity and quality-related characteristics of software components.
- **Interface Design Metrics**: Focus on the usability of the interface design.
- **Object-oriented Design Metrics**: Measure characteristics of classes, including their communication and collaboration.

Hierarchical Architecture Metrics

- **Fan out**: Number of modules immediately subordinate to a module.
- **Structural Complexity (S(i))**: [S(i) = f^2_{out}(i)]
- **Data Complexity (D(i))**: [D(i) = \frac{v(i)}{f_{out}(i) + 1}]
- **System Complexity (C(i))**: [C(i) = S(i) + D(i)]
- **Shape Complexity**:
  - Size: [\text{size} = n + a]
  - Connectivity Density (arc-to-node ratio): [r = \frac{a}{n}]

Metrics for Object-Oriented Design

- **Size**:
  - Population: Static count of all classes and methods.
  - Volume: Dynamic count of all instantiated objects at a given time.
  - Length: Depth of an inheritance tree.
- **Coupling**: Number of collaborations between classes or methods called between objects.
- **Cohesion**: Degree to which a class's properties are part of the problem or design domain.
- **Primitiveness**: Degree to which a method in a class is atomic.

Specific Class-oriented Metrics

- **Weighted Methods per Class**: Normalized complexity of the methods in a class.
- **Depth of the Inheritance Tree**: Maximum length from the derived class to the base class.
- **Number of Children**: As the number of children of a class grows, reuse increases but abstraction can be diluted, and testing effort increases.
- **Coupling Between Object Classes**: Measures the number of collaborations a class has with other classes.
- **Response for a Class**: Set of methods potentially executed in response to a public method call from outside the class.
- **Lack of Cohesion in Methods**: Measures the number of methods accessing the same instance variables.

Metrics for Source Code

- **Complexity Metrics**: Measure the logical complexity of source code.
- **Length Metrics**: Provide an indication of the size of the software.

Metrics for Testing

- **Statement and Branch Coverage Metrics**: Lead to the design of test cases that provide program coverage.
- **Defect-related Metrics**: Focus on defects found during testing.
- **Testing Effectiveness Metrics**: Provide a real-time indication of the effectiveness of tests conducted.
- **In-process Metrics**: Process-related metrics determined as testing is conducted.

Metrics for Maintenance

- **Software Maturity Index (SMI)**: [SMI = \frac{MT - (Fa + Fc + Fd)}{MT}]

  - (MT): Number of modules in the current release.

  - (Fa): Number of modules added.
  - (Fc): Number of modules changed.
  - (Fd): Number of modules deleted.

- **Factors Affecting Maintenance Costs**:

  - Team stability
  - Contractual responsibility
  - Staff skills
  - Program age and structure

- **Process Metrics for Maintenance**:

  - Number of requests for corrective maintenance
  - Average time for impact analysis
  - Average time to implement a change request
  - Number of outstanding change requests

Maintenance Cost Models

- **Belady and Lehman Model**:

  - Effort and cost can increase exponentially if a poor software development approach is used and the team that developed the software is not available for maintenance.
  - Basic equation: [M = P \times K \times c \times d]
    - (M): Total effort expended.
    - (P): Productive effort (analysis, design, coding, testing, evaluation).
    - (K): Empirically determined constant.
    - (c): Complexity measure due to lack of good design and documentation.
    - (d): Degree to which the maintenance team is familiar with the software.

- **Boehm Model**:

  - Annual Maintenance Effort (AME) is calculated as: [\text{AME} = \frac{P \times K}{1 - (E \times L)}]
    - (P): Product size.
    - (K): Complexity-adjustment factor.
    - (E): Environment factor.
    - (L): Learning factor.

Example Questions on Calculations

1. **Software Maturity Index (SMI)**:

   - Given:
     - (MT = 200)
     - (Fa = 10)
     - (Fc = 15)
     - (Fd = 5)
   - Calculate the Software Maturity Index (SMI).

2. **Belady and Lehman Model Calculation**:

   - Given:
     - (P = 100)
     - (K = 2)
     - (c = 1.5)
     - (d = 1.2)
   - Calculate the total effort (M).

3. **Boehm Model Calculation**:

   - Given:
     - (P = 200)
     - (K = 1.2)
     - (E = 0.3)
     - (L = 0.1)
   - Calculate the Annual Maintenance Effort (AME).