Software Quality Assurance: A Practical Perspective

Software Quality Assurance is:
- A group responsibility to support quality achievement in both product and process (the Stewards of Quality Commitment).
- Closely tied to verification and validation (V&V) conducted at each stage in the software life cycle.
- A management function.
- Maintenance of independence in relations with software development groups.
- Focused on process, but concerned with product quality.

Software Measurement

- Why?
  - Different goals
- What?
  - Software quality
- How?
  - Rationale for measurement
- When?
  - Throughout project development and product deployment

Software Measurement: Why?

- Goals of Software Measurement:
  - Increased productivity
    - Individual/group productive effort
    - Often related to short-term goals expressed in $ and days
  - Enhanced quality
    - Typically, quite subjective and controversial
    - Often related to long-term goals expressed in $ and months
Software Measurement: Why?

- Goals of Software Measurement:
  - Increased productivity
  - Enhanced quality
  - Improved process
    - The proper organizational view
    - Longer-term goals, not so easily expressed, either in $ or years

Focus: Software Quality

- What is Software Quality?
  - What is [high] quality software?
    - Is it software that
      - meets all requirements?
      - is produced within schedule?
      - is produced under cost?
      - has < 1 defect / 100 KSLOC?
      - has MTBF > 7200 hours?

Focus: Software Quality

- Without knowing more, the question is unanswerable.
- Additional knowledge is requisite in Objectives
  - (Objectives - Quality)

Software Measurement: What?

- What is Software Quality?
  - Software Quality: expressed in terms of project objectives
    - correctness
    - maintainability
    - reliability
    - reusability
    - testability
    - portability
    - adaptability
- Software quality is not expressed in terms of cost, scheduled completion time, or efficiency
- Cost, schedule, and efficiency are potential constraints on software quality

Software Measurement: How?

- Rationale for Software Quality Measurement
  - Objectives
    - Define
    - Prioritize
Software Measurement: How?

- Rationale for Software Quality Measurement
  - Objectives
  - Principles
    - Govern the process
    - Realize the objectives

Software Engineering Principles

- Principle (Webster’s)
  "A fundamental truth, law, doctrine or motivating force, upon which others are based"
  - Note the qualifier "fundamental"
    - Not simply a method or technique
    - Form cornerstones for a "methodology"

- Principles ⇐ Objectives

Software Engineering Principles

- Software Development methodology
  - Principles form the foundation

Software Development Process

- Software Development Methodology
- Principles

Software Measurement: How?

- Rationale for Software Quality Measurement
  - Objectives
  - Principles
  - Attributes
    - Desirable and beneficial
    - Recognizable
      - Products
      - Process

Software Engineering Attributes

- Attributes (Webster’s)
  "A characteristic or quality of a person or thing"
  Note: Implied is “identifiable”

- Attributes are induced by Principles
- Attributes are associated with Components

Software Engineering Attributes

- Attributes are evident in degrees
  - Can be present to high degree
  - Can be lacking or absent
Software Measurement: When?
Life-Cycle Basis for Software Quality Assessment

The OPA Framework exploits both Process and Product Indicators

- Early, no product deliverables exist
- Software Development Plan
- Methodology Manuals
- Transition from process to product emphasis

Presentation Focus
- Measurement of software quality, using the
- Rationale described above as Objectives/Principles/Attributes (OPA), relying on
- Indicators of software quality applied to components, enabling an
- Evaluation Procedure incorporating product (code and documentation) and process examination, that leads to
- Assessment and Prediction in quantitative terms meaningful to
- Software Engineers, Software Managers, and Project Managers

Elaborating on the OPA Rationale
- Any project must be undertaken with a set of objectives:
  - Objectives represent the desirable claims about the project in total
  - Explicit recognition and prioritization of objectives is preferable

Elaborating on the OPA Rationale
- Achieving those objectives requires a process governed by certain principles:
  - A development methodology should identify these principles
  - A development environment should supply tools supporting/mandating these principles

Elaborating on the OPA Rationale
- A process governed by these principles generates a product with desirable attributes:
  - Attributes are typically conceptual and resistant to direct measurement
  - Linkages among attributes/principles and principles/objectives are keys to measurement.

Primary Software Engineering Objectives
1. Adaptability
   the ease with which software can accommodate to changing requirements.
2. Correctness
   strict adherence to specifications.
3. Maintainability
   the ease with which corrections can be made to respond to recognized inadequacies.
Primary Software Engineering Objectives

4. Portability
   the ease in transferring software to another host environment.

5. Reliability
   the error-free behavior of software over time.

6. Reusability
   the use of software developed in other applications.

7. Testability
   the ability to evaluate conformance with specifications.

Primary Software Engineering Principles

1. Abstraction
   defining each program segment at a given level of refinement
   (a) Hierarchical Decomposition - components defined in a top-down manner.
   (b) Functional Decomposition - components partitioned along functional boundaries

2. Concurrent Documentation
   management of supporting documents (system specifications, user manuals, etc.) throughout the life cycle.

Primary Software Engineering Principles

3. Information Hiding
   insulating the internal details of component behavior.

4. Life Cycle Verification
   verification of requirements throughout the design, development, and maintenance phases of the life cycle.

5. Stepwise Refinement
   utilizing convergent design.

6. Structured Programming
   using a restricted set of program control constructs.

Objective/Principles Linkages

Adaptability
Correctness
Maintainability
Portability
Reliability
Reusability
Testability
Concurrent Documentation
Abstraction:
Decomposition,
Refinement,
Elaboration
Information Hiding
Life Cycle Verification
Stepwise Refinement
Structured Programming

Primary Software Engineering Attributes

1. Cohesion
   The binding of statements within a software component

2. Complexity
   an abstract measure of work associated with a software component

3. Coupling
   the interdependence among software components

Primary Software Engineering Attributes

4. Early Error Detection
   indication of faults in requirements, specification and design prior to implementation

5. Ease of Change
   software that accommodates enhancements or extensions

6. Readability
   the difficulty in understanding a software component
Primary Software Engineering Attributes

7. **Traceability**
   - the ease in retracing the complete history of a software component from its current status to its design

8. **Visibility of Behavior**
   - the provision of a review process for error checking

9. **Well-Defined Interfaces**
   - the definitional clarity and completeness of a shared boundary between software and/or hardware (software/software, software/hardware)

Principles/Attributes Linkages

- Concurrent Documentation
- Abstraction: Decomposition, Refinement, Elaboration
- Information Hiding
- Life Cycle Verification
- Stepwise Refinement
- Structured Programming
- Cohesion
- Complexity
- Coupling
- Early Error Detection
- Ease of Change
- Readability
- Traceability
- Visibility of Behavior
- Well-Defined Interfaces

OPA Framework for Software Development

- **OBJECTIVES**
- **PROCESS**
- **PRODUCT**
- **PRINCIPLES**
- **ATTRIBUTES**
- **DOCUMENTATION**
- **PROGRAMS**

Illustration of the Framework Linkages

Assessment and Prediction

- **Quantitative scale of [-5, 5]**
- - Accumulation of evidence - attribute presence or absence
- - Confirming and contrasting indicators

Assessment and Prediction

- **Assessment**: Entire development history through Integration Test is available
  - Can be applied to product only
  - Can be applied retrospectively

- **Prediction**: Partial development history - selected points
Establishing the Need for Software Quality Indicators

- Concepts elude measurement
  - Level of abstraction often too high
  - Requires subjective judgment

Measurement applied to product development

Characteristics of SQI’s

- SQI’s must be
  1. Measurable through the analysis of the software development process and products of that process, e.g. programs and documentation
  2. Indicative of the presence or absence of one or more attributes

- SQI’s can be
  1. “Raw statistics extracted from process activity, code and documentation analysis
  2. Variables computed using “raw” statistics

- SQI’s should be
  1. Simple, understandable, easily related to attribute(s)
  2. Targeted at design information (documentation) and at implementation (code and documentation)
  3. As objective as possible
The Foundations of Software Quality Indicators

Working Definition:
A software quality indicator (SQI) is a variable whose value can be determined through direct analysis of product or process characteristics and whose evidential relationship to one or more software engineering attributes is undeniable.

Software Engineering Attribute: an unmeasurable concept associated with software quality

Crucial Aspects:
1. Value is directly measurable, and
2. An SQI is always an attribute/variable pair

SQI Example: (CP/SDT)

Coupling relative to use of Structured Data Types (assumed scale: -5 to 5).

If (# of calls > 0) and (# of parms > 0) then

\[ \text{CP/SDT} := -5 \times \left( \frac{\# \text{ SDT parms}}{\# \text{ total parms}} \right) \]

else

\[ \text{CP/SDT} := 0 \]

Software Quality Indicators: Measurement Rationale (Process)

- Procedure

- Process Property:
- Impact of Property:
- OPA Entity Affected:
- Rationale:
- Measurement Approach:
- Metric(s):
- Indicator:

Software Quality Indicator: Process Example

- Process Property:
  - Development Instability
- Impact of Property:
  - Instability in development can stem from changes in:
    (1) personnel responsible for component
    (2) target hardware platform,
    (3) target language or language translator,
    (4) software development environment, and
    (5) project and organizational management or policy.

Software Quality Indicator: Process Example

- OPA Entity Affected:
  - Early Error Detection (-)
- Rationale:
  - Any form of instability can lead to discontinuities in the development process that:
    • cause an inordinate number of errors, making it difficult to detect and remove all of them, or
    • undermine and weaken the error detection capabilities resident in the process.
Software Quality Indicator:  
**Process Example**

- **Metric:**
  - Instability Factor = \[ \sum_{j=1}^{5} I(j) \]
  - \( I(j) \) =
    - 0 with no change of type \( j \)
    - 1 with one change of type \( j \)
    - 2 with two changes of type \( j \)
  - \( j = 1, \ldots, 5 \) from above

- **Indicator:**
  - Early Error Detection/Development Instability
  - EED/DI = 5 - Instability Factor

Software Quality Indicator:  
**ADA Code Example**

- **Rationale:**
  - Packages support the isolation and localization of related subprograms, and thereby foster logical, procedural and functional cohesion.

- **Measurement Approach:**
  - If the subprograms being exported from a package are related (functionally), then any units that “with” the package should use a substantial number of the exported subprograms. Hence, one approach to measuring the relatedness (cohesion) among the subprograms is to assess for each “with” the utilization of the subprogram set defined by the “withed” package.

- **Metric:**
  - Subprogram Utilization (Per Package Basis) = \[ \sum \frac{\text{"withs" to package}}{\text{# of referenced subprograms in package specification}} \]


**Evaluation in Meaningful Terms**

- Software Engineer \( \Rightarrow \) Attributes \( \Rightarrow \) Product
- Software Manager \( \Rightarrow \) Principles \( \Rightarrow \) Process
- Project Manager \( \Rightarrow \) Objectives \( \Rightarrow \) Project

Procedural synthesis conveys a consistent picture from one level to the next.
Decompositional trace points to specific problem needing corrective attention.