Chapter 21

- **Software Quality Assurance**

Slide Set to accompany

Software Engineering: A Practitioner's Approach

by Roger S. Pressman

Software Quality Assurance (SQA)

- Often called *quality management*
- An umbrella activity that is applied throughout the software process.

Table of Contents

21.1 Background Issues

- The first formal quality assurance and control function introduced at Bell Labs in 1916
- Spread rapidly throughout the manufacturing world.
- During the 1940s, more formal approaches to quality control were suggested.
- key elements:
  - Measurement
  - Continuous process improvement [Dem86]

21.2 Elements of SQA

- Standards
- Reviews and Audits
- Testing
- Error/defect collection and analysis
- Change management
- Education
- Vendor management
- Security management
- Safety
- Risk management

21.3 SQA Processes and Product Characteristics

- SQA procedures and approaches that work in one software environment may not work in another
  - Should understand the specific quality requirements for a software product and then select the process and specific SQA actions and tasks for those requirements
  - CMMI and ISO 9000 standards can be used

21.4 SQA Tasks, Goals, and Metrics

21.4.1 SQA Tasks

- **Prepares an SQA plan for a project.**
  - Identify
    - evaluations to be performed
    - audits and reviews to be performed
    - standards that are applicable to the project
    - procedures for error reporting and tracking
    - documents to be produced by the SQA group
    - amount of feedback provided to the software project team
- **Participates in development of the software process description.**
  - The SQA group reviews the process description for compliance with
    - organizational policy
    - internal software standards
    - externally imposed standards (e.g., ISO-9001)
    - other parts of the software project plan.

- **Reviews software engineering activities to verify compliance with the defined software process.**
  - identifies, documents, and tracks deviations from the process and verifies that corrections have been made.
- **Audits designated software work products to verify compliance with those defined as part of the software process.**
  - reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made
  - periodically reports the results of its work to the project manager.
- **Ensures that deviations in software work and work products are documented and handled according to a documented procedure.**
- **Records any noncompliance and reports to senior management.**
  - Noncompliance items are tracked until they are resolved.

21.4.2 Goals, Attributes and Metrics

- **Requirements quality**
  - The correctness, completeness, and consistency
  - have a strong influence on the quality of all work products
- **Design quality**
  - Every element of the design model should be assessed to ensure that it exhibits high quality and that the design itself conforms to requirements.
- **Code quality**
  - Source code and related work products (e.g., other descriptive information) must conform to local coding standards and exhibit characteristics that will facilitate maintainability.
- **Quality control effectiveness**
  - Should apply limited resources to have a high quality result.

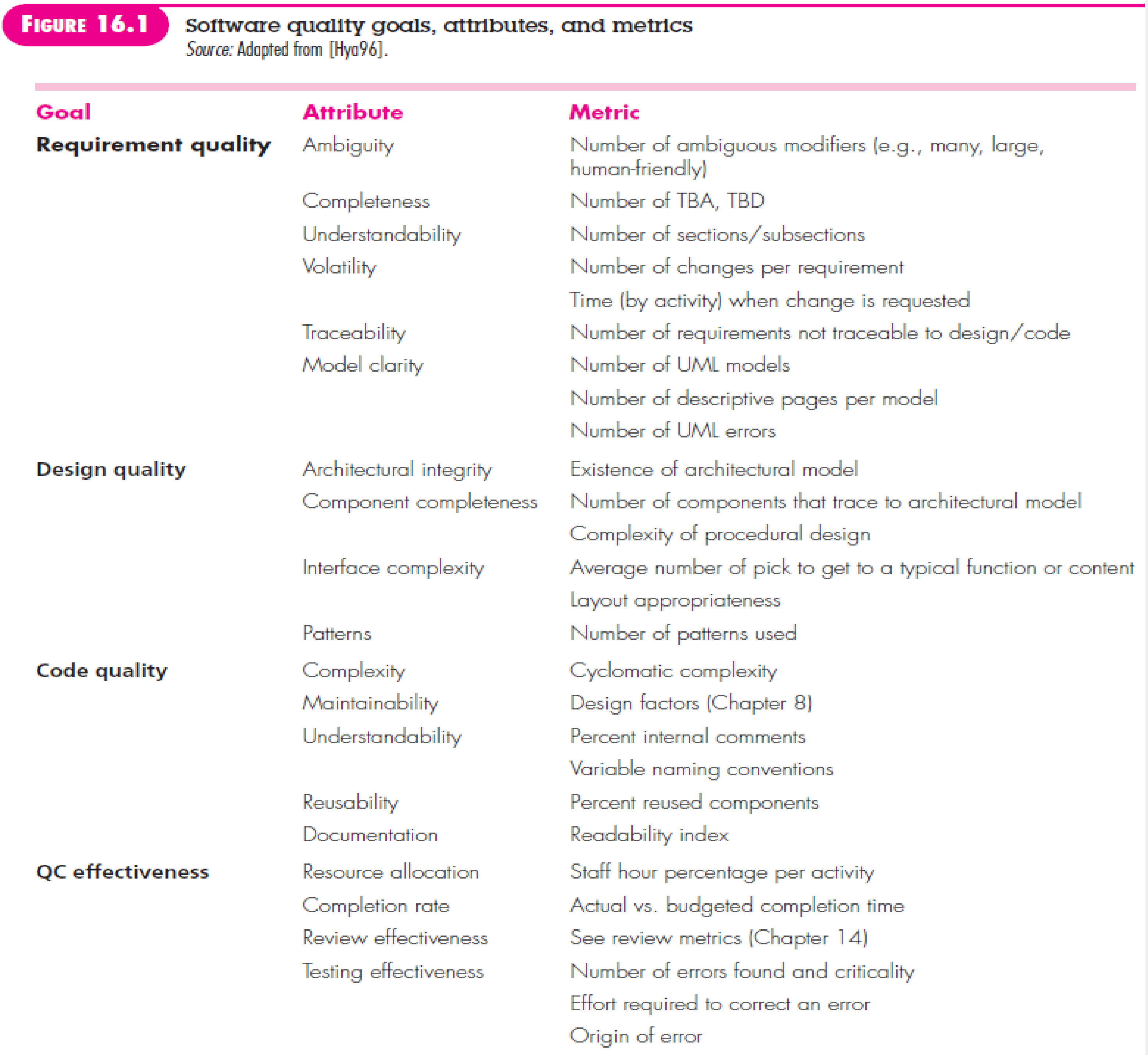


FIGURE 16.1 Software quality goals, attributes, and metrics

Source: Adapted from [Hya96].

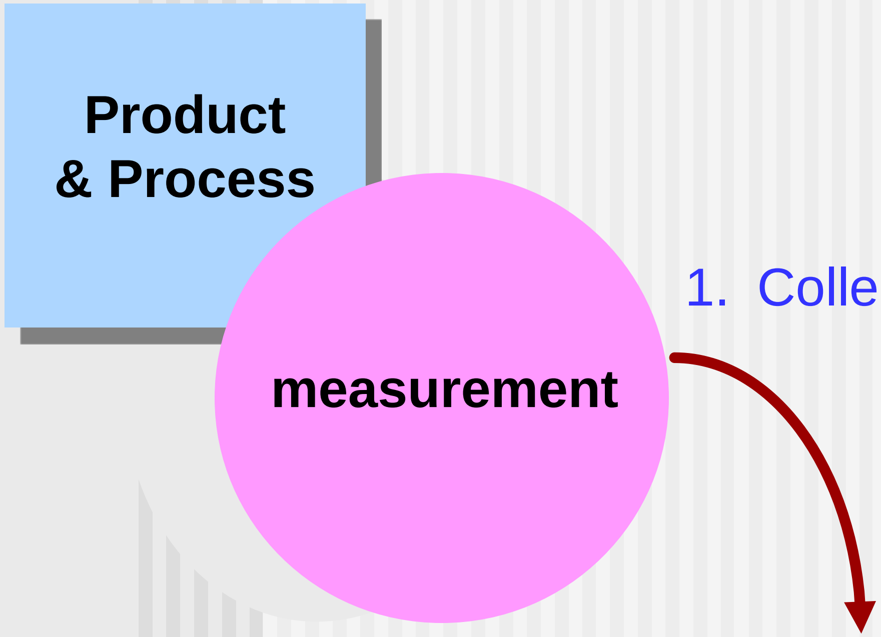
Goal	Attribute	Metric
Requirement quality	Ambiguity	Number of ambiguous modifiers (e.g., many, large, human-friendly)
	Completeness	Number of TBA, TBD
	Understandability	Number of sections/subsections
	Volatility	Number of changes per requirement
		Time (by activity) when change is requested
	Traceability	Number of requirements not traceable to design/code
	Model clarity	Number of UML models
		Number of descriptive pages per model
		Number of UML errors
Design quality	Architectural integrity	Existence of architectural model
	Component completeness	Number of components that trace to architectural model
		Complexity of procedural design
	Interface complexity	Average number of pick to get to a typical function or content
		Layout appropriateness
	Patterns	Number of patterns used
Code quality	Complexity	Cyclomatic complexity
	Maintainability	Design factors (Chapter 8)
	Understandability	Percent internal comments
		Variable naming conventions
	Reusability	Percent reused components
	Documentation	Readability index
QC effectiveness	Resource allocation	Staff hour percentage per activity
	Completion rate	Actual vs. budgeted completion time
	Review effectiveness	See review metrics (Chapter 14)
	Testing effectiveness	Number of errors found and criticality
		Effort required to correct an error
		Origin of error

21.5 Formal Approaches to SQA

- A computer program is a mathematical object.
- A rigorous syntax and semantics can be defined for every programming language, and a rigorous approach to the specification of software requirements (Ch. 21) is available.

☛ If so, then it should be possible to apply mathematic proof of correctness to demonstrate that a program conforms exactly to its specifications.

21.6 Statistical SQA



... an understanding of how to improve quality ...

1. Collect data on all defects
2. Find their causes
3. Provide fixes for the process

1. Data about software errors and defects is collected and categorized.
2. Trace each error and defect to its underlying cause:
   - E.g. non-conformance to specifications
     design error
     violation of standards
     poor communication with the customer).
   - Using the Pareto principle
     (80% of the defects can be traced to 20% of all possible causes),
     isolate the 20% (the *vital few*).
3. Correct the problems that have caused the errors and defects.

Taxonomy of Defect Causes

- Incomplete or erroneous specifications (IES)
- Misinterpretation of customer communication (MCC)
- Intentional deviation from specifications (IDS)
- Violation of programming standards (VPS)
- Error in data representation (EDR)
- Inconsistent component interface (ICI)
- Error in design logic (EDL)
- Incomplete or erroneous testing (IET)
- Inaccurate or incomplete documentation (IID)
- Error in programming language translation of design (PLT)
- Ambiguous or inconsistent human/computer interface (HCI)
- Miscellaneous (MIS)

21.5.1 A Generic Example

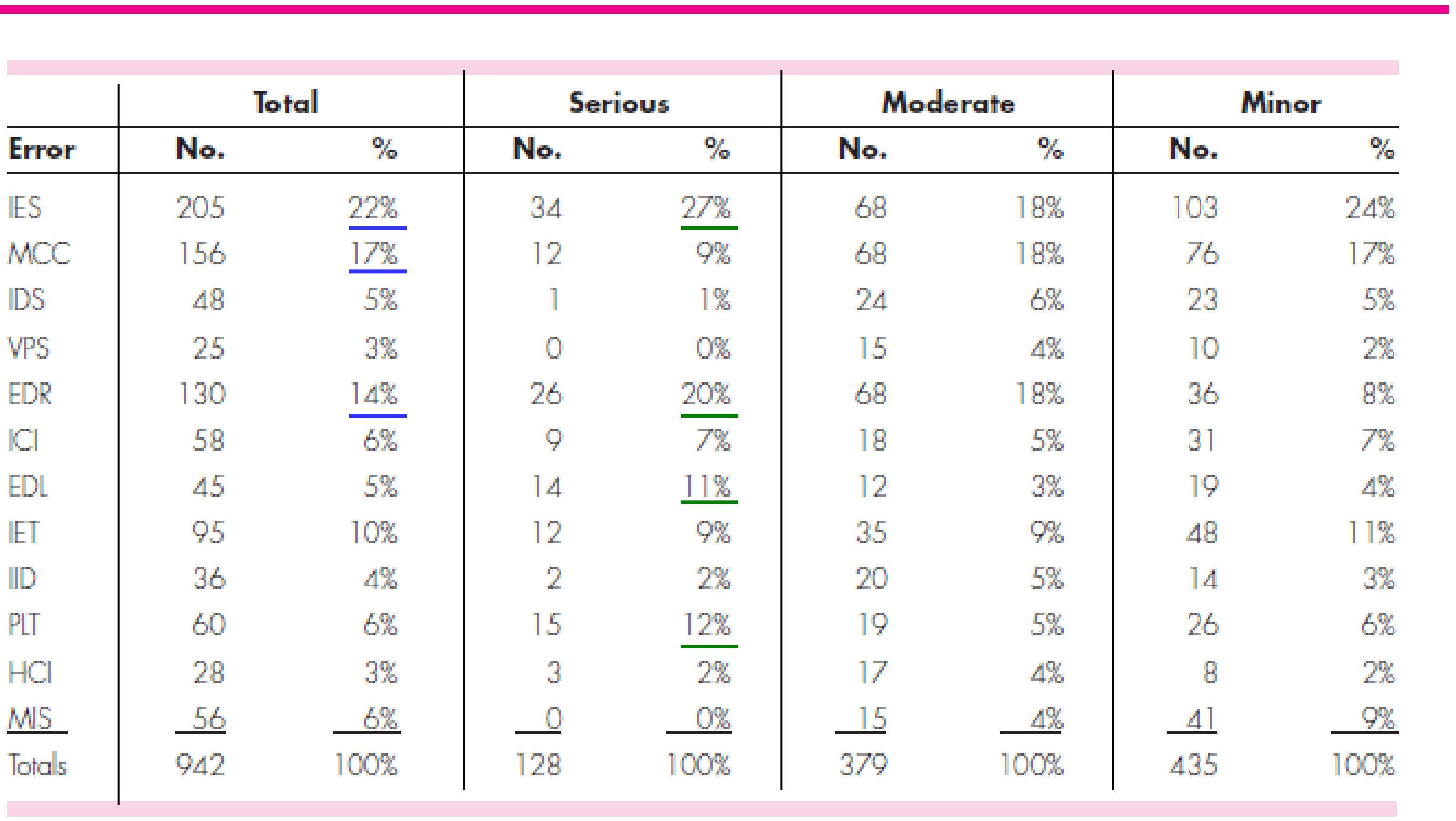


FIGURE 16.2

Data collection for statistical SQA

	Total		Serious		Moderate		Minor	
Error	No.	%	No.	%	No.	%	No.	%
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PLT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
Totals	942	100%	128	100%	379	100%	435	100%

Which are the vital few?

Which are the vital few of the serious errors?

What is the next step?

21.5.2 Six-Sigma for Software Engineering

- The most widely used strategy "… to measure and improve a company's operational performance by identifying and eliminating defects in manufacturing and service-related processes."
- The term "six sigma" is derived from six standard deviations (= σ)
  - 3.4 instances (defects) per million occurrences— an extremely high quality standard
- 3 core steps:
  - Define customer requirements and deliverables and project goals via well-defined methods of customer communication
  - Measure the existing process and its output to determine current quality performance (collect defect metrics)
  - Analyze defect metrics and determine the vital few causes.
- If an existing software process is in place, can have more steps
  - Improve the process by eliminating the root causes of defects.
  - Control the process to ensure that future work does not reintroduce the causes of defects.

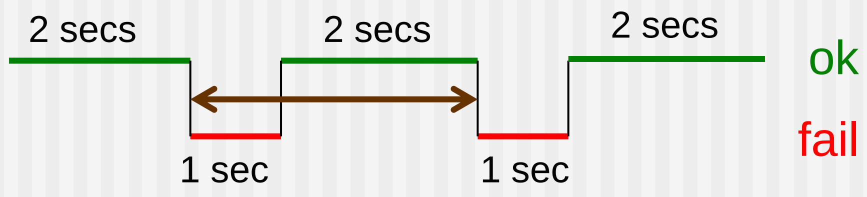
21.7 Software Reliability
21.7.1 Software Reliability and Availability

- Software reliability
  - "The probability of failure-free operation of a computer program in a specified environment for a specified time."
  - A simple measure of reliability is mean-time-between-failure (MTBF), where

MTBF = MTTF + MTTR

MTTF: mean-time-to-failure

MTTR: mean-time-to-repair



What is MTBF?

- Software availability
  - The probability that a program is operating according to requirements at a given point in time

Availability = [MTTF/(MTTF + MTTR)] x 100%

What is the availability of the system above?

21.7.2 Software Safety

- Software safety
  - Safeness from potential hazards that may affect software negatively and cause an entire system to fail. ☚ Narrow !
  - ☛ If hazards can be identified early,

    design features can be specified

    that will either eliminate or control potential hazards.
- However, according to [Leveson 11]
  - Hazard is “a system state or a set of conditions that, together with a particular set of worst-case environmental conditions, will lead to an accident (loss).” ☚ Safety-related !
  - Hazard + Environmental Conditions => Accident(Loss).

21.8 ISO 9001:2000 Standard

- Quality assurance standard that applies to software engineering.
- Contains 20 requirements that must be present for an effective quality assurance system.
- They address topics such as
  - management responsibility
  - quality system
  - contract review
  - design control
  - document and data control
  - product identification and traceability
  - process control
  - inspection and testing
  - corrective and preventive action
  - control of quality records
  - internal quality audits
  - training
  - servicing
  - statistical techniques.
- ISO 9000
  - Describes quality assurance elements in generic terms that can be applied to any business regardless of the products or services offered.

ISO 26262 -"Road vehicles – Functional safety"

- A functional safety standard.
- Functional safety:
  - "… freedom from unacceptable risk of physical injury or of damage to the health of people … "
  - An integral part of each automotive product development phase, ranging from the specification, to design, implementation, integration, verification, validation, and production release.
- Defines functional safety for automotive equipment applicable throughout the lifecycle of all automotive electronic and electrical safety-related systems.
- Can be used to set a barrier in automobile trade

21.9 The SQA Plan

- Provides a road map for instituting software quality assurance.
- Developed by the SQA group (or by the software team)
- Serves as a template for SQA activities