MODULE - V Software Quality Metrics - Objectives of quality measurement- Classification - Product metrics – Implementation – limitation; scope of quality management standards: ISO 9000 family, CMM and CMMI.

Software Quality Metrics

Two alternative but complementary definitions (IEEE, 1990) describe quality metrics as a category of SQA tools:

(1) A quantitative measure of the degree to which an item possesses a given quality attribute.

(2) A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute.

Experience with metrics in the field has not threatened its potential significance, attested to by the ISO 9000-3 standard (see ISO/IEC (1997) Sec. 4.20 and ISO (2001) Sec. 8). Software quality metrics are also an important part of the CMM guidelines (see Paulk *et al*., 1995).

Main objectives of software quality metrics

(1) To facilitate management control as well as planning and execution of the appropriate managerial interventions. Achievement of this objective is based on calculation of metrics regarding:
■ Deviations of actual functional (quality) performance from planned performance
■ Deviations of actual timetable and budget performance from planned performance.

(2) To identify situations that require or enable development or maintenance process improvement in the form of preventive or corrective actions introduced throughout the organization. Achievement of this objective is based on:
■ Accumulation of metrics information regarding the performance of teams, units, etc.Comparison provides the practical basis for management's application of metrics and for SQA improvement in general. The metrics are used for comparison of performance data with *indicators*, quantitative values such as:

■ Defined software quality standards
■ Quality targets set for organizations or individuals
■ Previous year's quality achievements
■ Previous project's quality achievements
■ Average quality levels achieved by other teams applying the same development tools in similar development environments
■ Average quality achievements of the organization
■ Industry practices for meeting quality requirements.

Software quality metrics – requirements

General requirements	Explanation
Relevant	Related to an attribute of substantial importance
Valid	Measures the required attribute
Reliable	Produces similar results when applied under similar conditions
Comprehensive	Applicable to a large variety of implementations and situations
Mutually exclusive	Does not measure attributes measured by other metrics

Operative requirements	Explanation
Easy and simple	The implementation of the metrics data collection is simple and is performed with minimal resources
Does not require independent data collection	Metrics data collection is integrated with other project data collection systems: employee attendance, wages, cost accounting, etc. In addition to its efficiency aspects, this requirement contributes to coordination of all information systems serving the organization
Immune to biased interventions by interested parties	In order to escape the expected results of the analysis of the metrics, it is expected that interested persons will try to change the data and, by doing so, improve their record. Such actions obviously bias the relevant metrics. Immunity (total or at least partial) is achieved mainly by choice of metrics and adequate procedures

Classification of software quality metrics
Software quality metrics can fall into a number of categories. Here we use a two-level system.
The first classification category distinguishes between life cycle and other phases of the software system:
- Process metrics, related to the software development process .
- Product metrics, related to software maintenance.

The second classification category refers to the subjects of the measurements:
- Quality
- Timetable
- Effectiveness (of error removal and maintenance services)
- Productivity.

A sizeable number of software quality metrics involve one of the two following measures for system size:
- **KLOC** – this classic metric measures the size of software by thousands of code lines. Application of metrics that include KLOC is limited to software systems developed using the same programming language or development tool.
- **Function points** – a measure of the development resources (human resources) required to develop a program, based on the functionality specified for the software system.

Process metrics
Software development process metrics can fall into one of the following categories:
- Software process quality metrics
- Software process timetable metrics
- Error removal effectiveness metrics
- Software process productivity metrics.

Software process quality metrics
Software process quality metrics may be classified into two classes:
- Error density metrics.
- Error severity metrics.

Software process timetable metrics
Software process timetable metrics may be based on accounts of success (completion of milestones per schedule) in addition to failure events (noncompletion per schedule). The TTO and ADMC metrics are based on data for all relevant milestones scheduled in the project plan. The metrics presented here are based on the two approaches illustrated in Table 21.3.

Table 21.3: Software process timetable metrics

Code	Name	Calculation formula
TTO	Time Table Observance	$TTO = \frac{MSOT}{MS}$
ADMC	Average Delay of Milestone Completion	$ADMC = \frac{TCDAM}{MS}$

Key:
- MSOT = milestones completed on time.
- MS = total number of milestones.
- TCDAM = total Completion Delays (days, weeks, etc.) for All Milestones. To calculate this measure, delays reported for all relevant milestones are summed up. Milestones completed on time or before schedule are considered "O" delays. Some professionals refer to completion of milestones before schedule as "minus" delays. These are considered to balance the effect of accounted-for delays (we might call the latter "plus" delays). In these cases, the value of the ADMC may be lower than the value obtained according to the metric originally suggested.

Software process productivity metrics

This group of metrics includes "direct" metrics that deal with a project's human resources productivity as well as "indirect" metrics that focus on the extent of software reuse. Software reuse substantially affects productivity and effectiveness.

Table 21.5: Process productivity metrics

Code	Name	Calculation formula
DevP	Development Productivity	$DevP = \frac{DevH}{KLOC}$
FDevP	Function point Development Productivity	$FDevP = \frac{DevH}{NFP}$
CRe	Code Reuse	$CRe = \frac{ReKLOC}{KLOC}$
DocRe	Documentation Reuse	$DocRe = \frac{ReDoc}{NDoc}$

Key:
- DevH = total working hours invested in the development of the software system.
- ReKLOC = number of thousands of reused lines of code.
- ReDoc = number of reused pages of documentation.
- NDoc = number of pages of documentation.

Product metrics

Product metrics refer to the software system's operational phase – years of regular use of the software system by customers, whether "internal" or "external" customers, who either purchased the software system or contracted for its development. In most cases, the software developer is requiredto provide customer service during the software's operational phase.

Customer services are of two main types:
- **Help desk services (HD)** – software support by instructing customers regarding the method of application of the software and solution of customer implementation problems. Demand for these services depends to a great extent on the quality of the user interface (its "user friendliness") as well as the quality of the user manual and integrated help menus.
- **Corrective maintenance services** – correction of software failures identified by customers/users or detected by the customer service team prior to their discovery by customers. The number of software failures and their density are directly related to software development quality. For completeness of information and better control of failure correction, it is recommended that all software failures detected by the customer service team be recorded as corrective maintenance calls.

HD metrics are based on all customer calls while corrective maintenance metrics are based on failure reports. Product metrics generally rely on performance records compiled during one year (or any other specified period of time). This policy enables comparisons of successive years in addition to comparisons between different units and software systems. The array of software product metrics presented here is classified as follows:
- HD quality metrics
- HD productivity and effectiveness metrics
- Corrective maintenance quality metrics
- Corrective maintenance productivity and effectiveness metrics.

It should be remembered that software maintenance activities include:

■ Corrective maintenance – correction of software failures detected during regular operation of the software.
■ Adaptive maintenance – adaptation of existing software to new customers or new requirements.
■ Functional improvement maintenance – addition of new functions to the existing software, improvement of reliability, etc.

HD quality metrics

The types of HD quality metrics discussed here deal with:
■ HD calls density metrics – the extent of customer requests for HD services as measured by the number of calls.
■ Metrics of the severity of the HD issues raised.
■ HD success metrics – the level of success in responding to these calls. A success is achieved by completing the required service within the time determined in the service contract

.

HD calls density metrics

This section describes six different types of metrics. Some relate to the number of the errors and others to a weighted number of errors. As for size/volume measures of the software, some use number of lines of code while others apply function points. The sources of data for these and the other metrics in this group are HD reports. Three HD calls density metrics for HD performance are presented in Table 21.6.

Table 21.6: HD calls density metrics

Code	Name	Calculation formula
HDD	HD calls Density	$HDD = \frac{NHYC}{KLMC}$
WHDD	Weighted HD calls Density	$WHDD = \frac{WHYC}{KLMC}$
WHDF	Weighted HD calls per Function point	$WHDF = \frac{WHYC}{NMFP}$

Key:
■ NHYC = number of HD calls during a year of service.
■ KLMC = thousands of lines of maintained software code.
■ WHYC = weighted HD calls received during one year of service.
■ NMFP = number of function points to be maintained.

Severity of HD calls metrics

The metrics belonging to this group of measures aim at detecting one type of adverse situation: increasingly severe HD calls. The computed results may contribute to improvements in all or parts of the user interface (its "user friendliness") as well as the user manual and integrated help menus.This metric, the **Average Severity of HD Calls (ASHC)**,refers to failures detected during a period of one year (or any portion thereof,as appropriate):

$$ASHC = \frac{WHYC}{NHYC}$$

where WHYC and NHYC are defind as in Table 21.6.

Success of the HD services

The most common metric for the success of HD services is the capacity to solve problems raised by customer calls within the time determined in the service contract (*availability*). Thus, the metric for success of HD services compares the actual with the designated time for provision of these services. For example, the availability of help desk (HD) services for an inventory management software package is defined as follows:

■ The HD service undertakes to solve any HD call within one hour.
■ The probability that HD call solution time exceeds one hour will not exceed 2%.
■ The probability that HD call solution time exceeds four working hours will not exceed 0.5%.

One metric of this group is suggested here, **HD Service Success (HDS):**

$$HDS = \frac{NHYOT}{NHYC}$$

where NHYOT = number of HD calls per year completed on time during one year of service.

HD productivity and effectiveness metrics

Productivity metrics relate to the total of resources invested during a specified period, while effectiveness metrics relate to the resources invested in responding to a HD customer call.

HD productivity metrics

HD productivity metrics makes use of the easy-to-apply KLMC measure of maintained software system's size (see Table 21.6) or according to functionpoint evaluation of the software system. Two HD productivity metrics are presented in Table 21.7.

Table 21.7: HD productivity metrics

Code	Name	Calculation formula
HDP	HD Productivity	$HDP = \frac{HDYH}{KLMC}$
FHDP	Function point HD Productivity	$FHDP = \frac{HDYH}{NMFP}$

Key:
- HDYH = total yearly working hours invested in HD servicing of the software system.
- KLMC and NMFP are as defined in Table 21.6.

HD effectiveness metrics

The metrics in this group refer to the resources invested in responding to customers' HD calls. One prevalent metric is presented here, **HD Effectiveness (HDE):**

$$HDE = \frac{HDYH}{NHYC}$$

where HDYH and NHYC are as defined in Tables 21.7 and 21.6 respectively.

Table 21.7: HD productivity metrics

Code	Name	Calculation formula
HDP	HD Productivity	$HDP = \frac{HDYH}{KLMC}$
FHDP	Function point HD Productivity	$FHDP = \frac{HDYH}{NMFP}$

Key:
- HDYH = total yearly working hours invested in HD servicing of the software system.
- KLMC and NMFP are as defined in Table 21.6.

Corrective maintenance quality metrics

Software corrective maintenance metrics deal with several aspects of the quality of maintenance services. A distinction is needed between software system failures treated by the maintenance teams and failures of the maintenance service that refer to cases where the maintenance failed to provide a repair that meets the designated standards or contract requirements. Thus, software maintenance metrics are classified as follows:

- **Software system failures density metrics** – deal with the extent of demand for corrective maintenance, based on the records of failures identified during regular operation of the software system.
- **Software system failures severity metrics** – deal with the severity of software system failures attended to by the corrective maintenance team.
- **Failures of maintenance services metrics** – deal with cases where maintenance services were unable to complete the failure correction on time or that the correction performed failed.

- **Software system availability metrics** – deal with the extent of disturbances caused to the customer as realized by periods of time where the services of the software system are unavailable or only partly available.

Software system failures density metrics

The software system failures density metrics presented here relate to the number and/or weighted number of failures.Three software system failures density metrics are presented in Table 21.8.

Table 21.8: Software system failures density metrics

Code	Name	Calculation formula
SSFD	Software System Failure Density	$SSFD = \frac{NYF}{KLMC}$
WSSFD	Weighted Software System Failure Density	$WSSFD = \frac{WYF}{KLMC}$
WSSFF	Weighted Software System Failures per Function point	$WSSFF = \frac{WYF}{NMFP}$

Key:
- NYF = number of software failures detected during a year of maintenance service.
- WYF = weighted number of yearly software failures detected during a year of maintenance service.
- KLMC = thousands of lines of maintained software code.
- NMFP = number of function points designated for the maintained software.

Software system failures severity metrics

Metrics of this group detect adverse situations of increasingly severe failures in the maintained software. Results may trigger retesting of all or parts of the software system.This metric, the **Average Severity of Software System Failures (ASSSF),** refers to software failures detected during a period of one year (or alternatively a half or a quarter of a year, as appropriate):

$$ASSSF = \frac{WYF}{NYF}$$

Failures of maintenance services metrics

As mentioned above, maintenance services can fail either because they were unable to complete the failure correction on time or when the correction performed failed and a repeated correction is required. The metrics presented here relate to the second type of maintenance failure. A customer call related to a software failure problem that was supposed to be solved after a previous call is commonly treated as a maintenance service failure. For practical purposes, many organizations limit the time frame for the repeat calls to three months, although the period can vary by type of failure or some other organizational criterion. The metric, **Maintenance Repeated repair Failure (MRepF)**, is defined as follows:

$$MRepF = \frac{RepYF}{NYF}$$

where RepYF is the number of repeated software failure calls (service failures).

Software system availability metrics

User metrics distinguish between:
- Full availability – where all software system functions perform properly
- Vital availability – where no vital functions fail (but non-vital functions may fail)
- Total unavailability – where all software system functions fail.

The source for all availability metrics is user failure records. The latter specify the extent of damage (non-vital failure, vital failure and total system failure) as well as duration (hours) for each failure. Three software system availability metrics are presented in Table 21.9.

Table 21.9: Software system availability metrics

Code	Name	Calculation formula
FA	Full Availability	$FA = \frac{NYSerH - NYFH}{NYSerH}$
VitA	Vital Availability	$VitA = \frac{NYSerH - NYVitFH}{NYSerH}$
TUA	Total Unavailability	$TUA = \frac{NYTFH}{NYSerH}$

Software corrective maintenance productivity and effectiveness metrics

While corrective maintenance productivity relates to the total of human resources invested in maintaining a given software system, corrective maintenance effectiveness relates to the resources invested in correction of a single failure. Three software corrective maintenance productivity and effectiveness metrics are presented in Table 21.10.

Table 21.10: Software corrective maintenance productivity and effectiveness metrics

Code	Name	Calculation formula
CMaiP	Corrective Maintenance Productivity	$CMaiP = \frac{CMaiYH}{KLMC}$
FCMP	Function point Corrective Maintenance Productivity	$FCMP = \frac{CMaiYH}{NMFP}$
CMaiE	Corrective Maintenance Effectiveness	$CMaiE = \frac{CMaiYH}{NYF}$

Key:
- CMaiYH = total yearly working hours invested in the corrective maintenance of the software system.
- KLMC = thousands of lines of maintained software code.
- NMFP = number of function points designated for the maintained software.
- NYF = number of software failures detected during a year of maintenance service.

Implementation of software quality metrics
The application of software quality metrics in an organization requires:

- Definition of software quality metrics – relevant and adequate for teams, departments, etc.
- Regular application by unit, etc.
- Statistical analysis of collected metrics data
- Subsequent actions:

– Changes in the organization and methods of software development and maintenance units and/or any other body that collected the metrics data
– Change in metrics and metrics data collection
– Application of data and data analysis to planning corrective actions for all the relevant units.

Definition of new software quality metrics

The definition of metrics involves a four–stage process:
(1) Definition of attributes to be measured: software quality, development team productivity, etc.
(2) Definition of the metrics that measure the required attributes and confirmation of its adequacy in complying with the requirements
(3) Determination of comparative target values based on standards, previous year's performance, etc. These values serve as *indicators* of whether the unit measured (a team or an individual or a portion of the software) complies with the characteristics demanded of a given attribute.
(4) Determination of metrics application processes: – Reporting method, including reporting process and frequency of reporting
– Metrics data collection method.
The new metrics (updates, changes and revised applications) will be constructed following analysis of the metrics data as well as developments in the organization and its environment. The software quality metrics definition process is described in Figure 21.1.

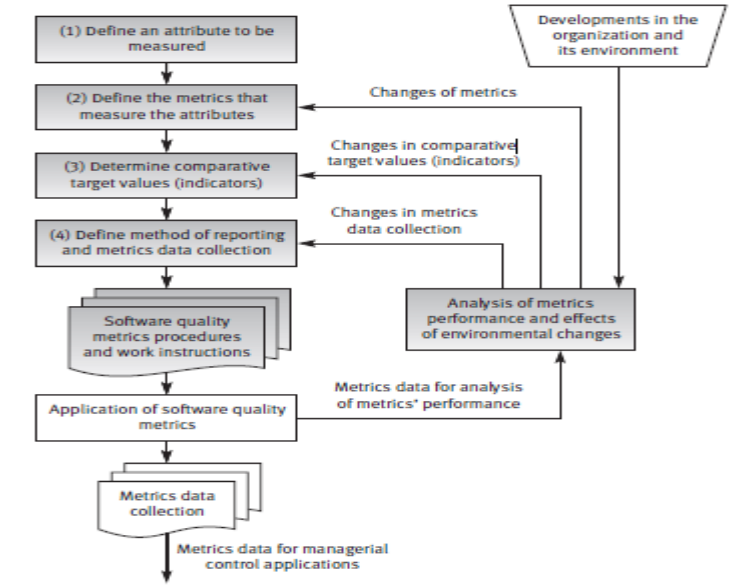

Figure 21.1: The process of defining software quality metrics

Application of the metrics – managerial aspects

The process of applying a metric or a set of metrics is similar to the implementation of new procedures or methodologies. It involves:

- Assigning responsibility for reporting and metrics data collection.
- Instruction of the team regarding the new metrics.
- Follow-up includes:

– Support for solving application problems and provision of supplementary information when needed.
– Control of metrics reporting for completeness and accuracy.

- Updates and changes of metrics definitions together with reporting and data collection methods according to past performance.

Example – Comparison of US and Japanese software industries

Cusumano (1991) makes use of three metrics in a comparison of the US and Japanese software industries:

- Mean productivity
- Failure density (based on measurements during the first 12 months after system delivery)
- Code reuse.

These metrics are presented in Table 21.11, and Cusumano's results are presented in Table 21.12.

Table 21.11: US and Japanese software industries – metrics

Name	Calculation formula
Mean productivity (similar to DevP, Table 21.5)	$\frac{KNLOC}{WorkY}$
Failure density (similar to SSFD, Table 21.8)	$\frac{NYF}{KNLOC}$
Code reuse (similar to CRe, Table 21.5)	$\frac{ReKNLOC}{KNLOC}$

Key:
- KNLOC – thousands of non-comment lines of code.
- WorkY – human work-years invested in the software development.
- ReKNLOC – thousands of reused non-comment lines of code.

Source: Based on Cusumano (1991)

Table 21.12: US and Japanese software industries – comparison of three software quality metrics

Software quality metrics	United States	Japan
Mean productivity	7290	12447
Failure density	4.44	1.96
Code reuse	9.71%	18.25%
N – (number of companies)	20	11

Source: Cusumano (1991)

Statistical analysis of metrics data

Analysis of metrics data provides opportunities for comparing a series of project metrics. These certainly include comparison of metrics results against predefined indicators, as well as comparisons with former projects or team performance at different periods of time, and so on. Another important comparison relates to the effectiveness with which the metrics themselves fulfil their respective aims.

Statistical tools provide us with two levels of support, based on the type of statistics used:
- Descriptive statistics
- Analytical statistics.

Descriptive statistics

Descriptive statistics, such as the mean, median and mode as well as use of graphic presentations such as histograms, cumulative distribution graphs, pie charts and control charts (showing also the indicator values) to illustrate the information to which they relate, enable us to quickly identify trends in the metrics values.

Analytical statistics

Description is not always enough. To determine whether the observed changes in the metrics are meaningful, whatever the direction, the observed trends must be assessed for their significance. This is the role of analytic statistics (e.g., regression tests, analysis of variance, or more basic tests such as the T-test and Chi-square test).

Taking action in response to metrics analysis results

The actions taken in response to metrics analysis can be classified as direct actions if initiated by the project or team management or indirect actions if initiated by the Corrective Action Board. The CAB indirect actions are a result of analysis of metrics data accumulated from a variety of projects and/or development departments.

Examples of the direct changes initiated by management include reorganization, changes in software development and maintenance methods, and revision of the metrics computed. For a comprehensive discussion of indirect actions as initiated by the Corrective Action Board.

Limitations of software metrics

Application of quality metrics is strewn with obstacles. These can be grouped as follows:
- Budget constraints in allocating the necessary resources (manpower, funds, etc.) for development of a quality metrics system and its regular application.
- Human factors, especially opposition of employees to evaluation of their activities.
- Uncertainty regarding the data's validity, rooted in partial and biased reporting.

The main factors affecting development proces parameters, especially their magnitude, are:

(1) Programming style: strongly affects software volume, where "wasteful" coding may double the volume of produced code (KLOC).

(2) Volume of documentation comments included in the code: affects volume of the code. The volume of comments is usually determined by the programming style (KLOC).

(3) Software complexity: complex modules require much more development time (per line of code) in comparison to simple modules. Complex modules also suffer from more defects than simple modules of similar size (KLOC, NCE).

(4) Percentage of reused code: the higher the percentage of reused code incorporated into the software developed, the greater the volume of code that can be produced per day as well as the lower the number of defects detected in reviews, testing and regular use (NDE, NCE).

(5) Professionalism and thoroughness of design review and software testing teams: affects the number of defects detected (NCE).

(6) Reporting style of the review and testing results: some teams produce concise reports that present the findings in a small number of items (small NCE), while others produce comprehensive reports, showing the same findings for a large number of items (large NDE and NCE).

The main factors affecting the magnitude of the product (maintenance) parameters are:

(1) Quality of installed software and its documentation (determined by the quality of the development team as well as the review and testing teams): the lower the initial quality of the software, the greater the anticipated software failures identified and subsequent maintenance efforts (NYF,NHYC).

(2) Programming style and volume of documentation comments included in the code: as in the development stage, both strongly affect the volume of the software to be maintained, where wasteful coding and documentation may double the volume of code to be maintained (KLMC).

(3) Software complexity: complex modules require investment of many more maintenance resources per line of code than do simple modules, and suffer from more defects left undetected during the development stage (NYF).

(4) Percentage of reused code: the higher the percentage of reused code, the lower the number of defects detected in regular use as well as the fewer required corrective maintenance and HD efforts (NYF).

(5) Number of installations, size of the user population and level of applications in use: affect the number of HD calls as well as the number of defects detected by users during regular use (NHYC, NYF). A unique difficulty faced by use of software quality metrics is rooted in the measures (parameters) that comprise many software quality metrics. As a result, a large proportion of software metrics, including most of the commonly used metrics, suffer from low validity and limited comprehensiveness. Examples of metrics that exhibit severe weaknesses are:
- Software development metrics that are based on measures such as KLOC, NDE and NCE

■ Product (maintenance) metrics that are based on measures such as KLMC, NHYC and NYF.
For example, the KLOC measure is affected by the programming style, the volume of documentation comments included in the code and the complexity of the software.
NYF is affected by the quality of the installed software and its documentation as well as the percentage of reused code, among the other factors affecting maintenance.

Scope of quality management standards: ISO 9000 family, CMM and CMMI.

The scope of quality management standards

Certification standards vary from assessment standards by content as well as by emphasis.The scope of certification standards is determined by the aims of certification,which are to:
■ Enable a software development organization to demonstrate consistent ability to assure that its software products or maintenance services comply with acceptable quality requirements. This is achieved by certification granted by an external body.
■ Serve as an agreed basis for customer and supplier evaluation of the supplier's quality management system. This may be accomplished by customer performance of a quality audit of the supplier's quality management system.The audit will be based on the certification standard's requirements.
■ Support the software development organization's efforts to improve quality management system performance and enhance customer satisfaction through compliance with the standard's requirements. The scope of assessment standards is also determined by the aims fo assessment,which are to:
■ Serve software development and maintenance organizations as a tool for self-assessment of their ability to carry out software development projects.
■ Serve as a tool for improvement of development and maintenance processes. The standard indicates directions for process improvements.
■ Help purchasing organizations determine the capabilities of potential suppliers.
■ Guide training of assessors by delineating qualifications and training program curricula.

ISO 9001 and ISO 9000-3
ISO 9000-3, the Guidelines offered by the International Organization for Standardization (ISO), represent implementation of the general methodology of quality management ISO 9000 Standards to the special case of software development and maintenance. Both ISO 9001 and ISO 9000-3 are reviewed and updated once every 5–8 years, with each treated separately. As ISO 9000-3 adaptations are based on those introduced to ISO 9001, publication of the revised Guidelines follows publication of the revised Standard by a few years. For example, the 1997 edition of ISO 9000-3 (ISO, 1997) relies on the 1994 edition of ISO 1994 (ISO, 1994). At the time of writing, the 2000 edition of ISO 9001 (ISO, 2000a) has been issued, but only the final just-completed draft of ISO 9000-3 (ISO/IEC, 2001) is awaiting approval.
The current 1997 edition of ISO 9000-3 Guidelines integrates ISO 9001 with its specialized ISO 9000-3 Guidelines into one "all inclusive" standard for the software industry. In other words, from the 1997 edition on, the ISO 9000-3 will represent the stand-alone ISO standard for the software industry.The new version of ISO 9000-3 follows this lead and will also serve as an "all-inclusive" standard for the software industry. Hence, the ISO 9000-3 Standard for the software industry can be considered to provide the requirements for ISO 9000-3 certification. The new ISO/IEC 9000-3 version (expected to be issued in 2003) is planned to serve the entire population of software development and maintenance organizations by adopting a policy of comprehensiveness and standard redundancy. The individual user is expected to tailor the standard to specific needs. These features facilitate achievement of the universality that allows ISO/IEC 9000-3 to fit the immense variety of organizations belonging to the software industry: big or small, developers of tailor-made software or COTS software packages, developers of real-time application software, embedded software or management information systems, etc. The 2000 edition of ISO 9001 as well as the new edition of ISO 9000-3 are supported by two additional conceptual standards: ISO 9000 (ISO, 2000b), which deals with fundamental concepts and terminology, and ISO 9004 (ISO, 2000c), which provides guidelines for performance improvement.
In the following sections, the principles underlying ISO 9000-3 are reviewed; in addition, the structure of the new version is comparedwith that of the current versions to illuminate their expanded applications. The last part of this section is dedicated to TickIT, an organization that significantly contributed to the adoption of ISO 9000-3.
ISO 9000-3 quality management system: guiding principles
Eight principles guide the new ISO 9000-3 standard; these were originally set down in the ISO 9000:2000 standard (ISO, 2000b), as follows:

(1) **Customer focus**. Organizations depend on their customers and therefore should understand current and future customer needs.
(2) **Leadership.** Leaders establish the organization's vision. They should create and maintain an internal environment in which people can become fully involved in achieving the organization's objectives via the designated route.
(3) **Involvement of people.** People are the essence of an organization; their full involvement, at all levels of the organization, enables their abilities to be applied for the organization's benefit.
(4) **Process approach**. A desired result is achieved more efficiently when activities and resources are managed as a process.
(5) **System approach to management**. Identifying, understanding and managing processes, if viewed as a system, contributes to the organization's effectiveness and efficiency.
(6) **Continual improvement**. Ongoing improvement of overall performance should be high on the organization's agenda.
(7) **Factual approach to decision making.** Effective decisions are based on the analysis of information.
(8) **Mutually supportive supplier relationships.** An organization and its suppliers are interdependent; a mutually supportive relationship enhances the ability of both to create added value.

ISO 9000-3: requirements

The current standard edition of ISO, 9000-3 (ISO 1997) includes 20 requirements that relate to the various aspects of software quality management systems. The new ISO 9000-3 (ISO/IEC, 2001) offers a new structure, with its 22 requirements classified into the following five groups:

- Quality management system
- Management responsibilities
- Resource management
- Product realization
- Management, analysis and improvement.

The new structure is presented in Table 23.1. The new structure realizes a change in emphasis among the various subjects that make up the requirements, a totally new classification of SQA topics into standard sections and revision of requirement section titles. These changes reflect a gradual rather than a radical change of concepts as presented in the updated guiding principles. Table 23.2 compares ISO 9000-3:1997 edition with those of the upcoming edition for a sample of requirement subjects, one for each requirement class.

Table 23.1: ISO 9000-3 new edition – Requirements and their classification

Requirement class	Requirement subjects
4. Quality management system	4.1 General requirements 4.2 Documentation requirements
5. Management responsibilities	5.1 Management commitments 5.2 Customer focus 5.3 Quality policy 5,4 Planning 5.5 Responsibility, authority and communication 5.6 Management review
6. Resource management	6.1 Provision of resources 6.2 Human resources 6.3 Infrastructure 6.4 Work environment
7. Product realization	7.1 Planning of product realization 7.2 Customer-related processes 7.3 Design and development 7.4 Purchasing 7.5 Production and service provision 7.6 Control of monitoring and measuring devices
8. Measurement, analysis and improvement	8.1 General 8.2 Monitoring and measurement 8.3 Control of non-conforming product 8.4 Analysis of data 8.5 Improvement

Source: ISO (2000a)

Table 23.2: Current ISO 9000-3:1997 vs. new edition – requirements comparison (sample)

ISO 9000-3: new edition Requirement class and subject	ISO 9000-3:1997 edition Requirement subjects
Requirement class 4 (Quality management system) Subject 4.2 Documentation requirements	4.2 Quality system 4.5 Document and data control 4.16 Control of quality records
Requirement class 5 (Management responsibilities) Subject 5.4 Planning	4.1 Management responsibility 4.2 Quality system
Requirement class 6 (Resource management) Subject 6.3 Infrastructure	4.9 Process control
Requirement class 7 (Product realization) Subject 7.5 Production and service provision	4.7 Control of customer-supplied product 4.8 Product identification and traceability 4.9 Process control 4.10 Inspection and testing 4.12 Inspection and test status 4.15 Handling, storage, packaging, preservation and delivery 4.19 Servicing
Requirement class 8 (Measurement, analysis and improvement) Subject 8.3 Control of non-conforming product	4.13 Control of non-conforming product

Source: Adapted from ISO (2000a)

ISO 9001 — application to software: the TickIT initiative

TickIT was launched in the late 1980s by the UK software industry in cooperation with the UK Department for Trade and Industry to promote development of a methodology for adapting ISO 9001 to the characteristics of the software industry known as the *TickIT initiative*. At the time of its launch, ISO 9001 had already been successfully applied in manufacturing industry; however, no significant methodology for its application to the special characteristics of the software industry was yet available. In the years to follow, the TickIT initiative, together with the efforts invested in development of ISO 9000-3, achieved this goal.

TickIT activities include:

- Publication of the *TickIT Guide*, that supports the software industry's efforts to spread ISO 9001 certification. The current guide (edition 5.0, TickIT, 2001), which includes references to ISO/IEC 12207 and ISO/IEC 15504, is distributed to all TickIT customers.
- Performance of audit-based assessments of software quality systems and consultation to organizations on improvement of software development and maintenance processes in addition to their management.
- Conduct of ISO 9000 certification audits.

Certification according to ISO 9000-3

The ISO 9000-3 certification process verifies that an organization's software development and maintenance processes fully comply with the standard's requirements.

As ISO 9000 standards have been adopted as national standards in many countries, there is growing worldwide interest in certification according to ISO 9000 by organizations in many industries, including the software industry. The certification service is organized by the International Organization for Standardization (ISO) through a worldwide network of certification services that are authorized by means of *accreditation bodies and certification bodies*. Each accreditation body is licensed by ISO to authorize other professional organizations as certification bodies. Certification bodies, whose number may vary by country, perform the actual certification audits and certify those organizations that qualify.

Organizations wishing to obtain ISO 9000-3 certification are required to complete the following:

- Develop the organization's SQA system
- Implement the organization's SQA system
- Undergo certification audits.

Fulfillment of these requirements demands thorough planning of the structures and resources necessary to perform the activities culminating in certification. This process may vary somewhat from one organization to another, depending on the characteristics of its design and maintenance activities as well as by the certification bodies. Its basic form parallels the process demanded by other certification standards. Certification is discussed in greater detail in the next four sections and is illustrated in Figure 23.1.

Planning the process leading to certification

Once management has made its decision to obtain ISO 9000-3 certification for its software development and maintenance activities, an action plan is needed.

An internal survey of the current SQA system and how it is implemented is a good place to begin. The survey should supply information about:

- Gaps between currently employed SQA and required procedures: missing procedures in addition to inadequate procedures.

■ Gaps between staff know-how and knowledge required regarding SQA procedures and SQA tools.
■ Gaps regarding documentation of development as well as maintenance activities.
■ Gaps or lack of parity regarding software configuration system capabilities and implementation.
■ Gaps regarding managerial practices demanded for project progress control.
■ Gap regarding SQA unit organization and its capabilities.

After completing the previous analysis, the plan for obtaining certification can be constructed. It should include:

■ A list of activities to be performed, including timetables
■ Estimates of resources required to carry out each activity
■ Organizational resources: (a) internal participants – SQA unit staff (including staff to be recruited) and senior software engineers; (b) SQA consultants.

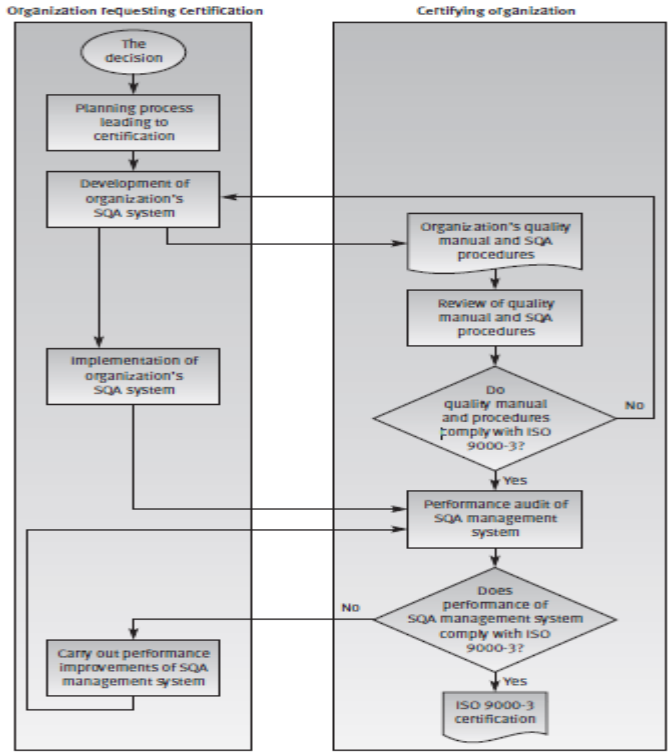

Figure 23.1: The ISO 9000-3 certification process

Development of the organization's SQA system

Before proceeding, the organization's SQA management system should be developed to a level adequate to meet ISO 9000-3 requirements. These efforts should include:

■ Development of a quality manual and a comprehensive set of SQA procedures.
■ Development of other SQA infrastructure:
– Staff training and instruction programs, including staff certification programs
– Preventive and corrective actions procedures, including the CAB committee
– Configuration management services, including a software change control management unit
– Documentation and quality record controls.
■ Development of a project progress control system.

Implementation of the organization's SQA system

Once the components of the SQA management system conform to certification demands, efforts are shifted towards implementing the system. These include setting up a staff instruction program and support services appropriate to the task of solving problems that may arise when implementing SQA tools. These arrangements are targeted especially at team leaders and unit managers, who are expected to follow up and support the implementation efforts made by their units.Throughout this stage, internal quality audits are carried out to verify the success in implementation as well as to identify units and SQA issues that require additional attention. The internal quality audit findings will enable determination of whether the organization has reached a satisfactory level of implementation.

Undergoing the certification audits

The certification audits are carried out in two stages:

(1) *Review of the quality manual and SQA procedures developed by the organization.* The review ascertains completeness and accuracy. In cases of non-compliance with standards, the organization is obligated to complete the corrections prior to advancing to the second stage of certification.

(2) *Verification audits of compliance with the requirements defined by the organization in its quality manual and SQA procedures.*

Procedures for retaining ISO certification

Periodic re-certification audits, usually carried out once or twice a year, are performed to verify continued compliance with ISO 9000-3 requirements. During these audits, the organization has to demonstrate continuing development of its SQA management system, which is expressed in quality and productivity performance improvements, regular updates of procedures to reflect technological changes, and process improvements.

Capability Maturity Models – CMM and CMMI assessment methodology

Carnegie Mellon University's Software Engineering Institute (SEI) took the initial steps toward development of what is termed a *capability maturity model* (CMM) in 1986, when it released the first brief description of the maturity process framework. The initial version of the CMM was released in 1992, mainly for receipt of feedback from the software community. The first version for public use was released in 1993 (Paulk *et al.*, 1993, 1995; Felschow, 1999).

The principles of CMM

CMM assessment is based on the following concepts and principles:

■ Application of more elaborate management methods based on quantitative approaches increases the organization's capability to control the quality and improve the productivity of the software development process.

■ The vehicle for enhancement of software development is composed of the five-level capability maturity model. The model enables an organization to evaluate its achievements and determine the efforts needed to reach the next capability level by locating the process areas requiring improvement.

■ Process areas are generic; they define the "what", not the "how". This approach enables the model to be applied to a wide range of implementation organizations because:

– It allows use of any life cycle model

– It allows use of any design methodology, software development tool and programming language

– It does not specify any particular documentation standard. The CMM and its key process areas (KPAs) are presented in Figure 23.2.

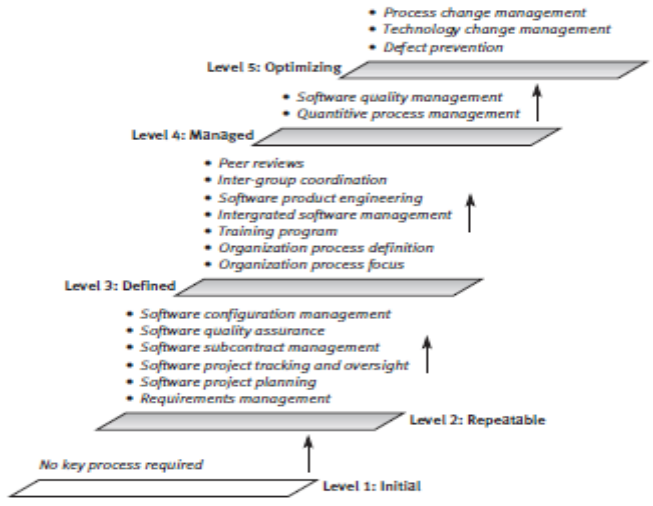

Figure 23.2: The CMM model levels and key process areas (KPAs)

Source: After Paulk *et al.* (1995)

The evolution of CMM

After 1993, the SEI expanded the original Software Development and Maintenance Capability Maturity Model (SW-CMM) through diversification. Its main structure was retailored to fit a variety of specialized capability maturity models. The following variants have been developed:

■ **System Engineering CMM (SE-CMM)** focuses on system engineering practices related to product-oriented customer requirements. It deals with product development: analysis of requirements, design of product systems, management and coordination of the product systems and their integration. In addition, it deals with the production of the developed product: planning production lines and their operation.

■ **Trusted CMM (T-CMM)** was developed to serve sensitive and classified software systems that require enhanced software quality assurance.

■ **System Security Engineering CMM (SSE-CMM)** focuses on security aspects of software engineering and deals with secured product development processes, including security of development team members.

■ **People CMM (P-CMM)** deals with human resource development in software organizations: improvement of professional capacities, motivation,organizational structure, etc.
■ **Software Acquisition CMM (SA-CMM)** focuses on special aspects of software acquisition by treating issues – contract tracking, acquisition risk management, quantitative acquisition management, contract performance management, etc. – that touch on software purchased from external organizations.
■ **Integrated Product Development CMM (IPD-CMM)** serves as a framework for integration of development efforts related to every aspect of the product throughout the product life cycle as invested by each department. Practically speaking, this CMM overlaps key processes of SW-CMM and SE-CMM rather considerably, hence its elements were integrated into a CMMI model (see the following) and its development was discontinued.

Capability Maturity Model Integration (CMMI)

In the late 1990s a new developmental direction was taken – development of integrated CMM models. Development of specialized CMM models involved development of different sets of key processes for model variants for different departments that exhibited joint processes. In practice, this created a situation where departments that applied different CMM variants in the same organization faced difficulties in cooperation and coordination. The CMMI approach solved these problems at the same time as the moduals better conformed to the emerging ISO/IEC 15504 standard (see Royce, 2002). At the beginning of 2002, SEI could offer the 1.1 version of three CMMI models, with each model presenting different integrated components:
■ CMMI-SE/SW integrates the system engineering and software engineering .
■ CMMI-SE/SW/IPPD/SS integrates system engineering, software engineering and integrated product/process and supplier sourcing engineering aspects.
■ CMMI-SE/SW/IPPD integrates system engineering, software, integrated product/process and supplier sourcing aspects.

The CMMI structure and processes areas

The CMMI model, like the original CMM models, is composed of five levels. The CMMI capability levels are the same as those of the original, apart from a minor change related to capability level 4, namely:
■ Capability maturity level 1: Initial
■ Capability maturity level 2: Managed
■ Capability maturity level 3: Defined
■ Capability maturity level 4: Quantitatively managed
■ Capability maturity level 5: Optimizing.

A substantial change has nonetheless evolved with respect to the processes included in the models. The 18 key process areas of CMM (frequently referred to as *KPAs*) were replaced by 25 process areas (PAs). The PAs are classified by the capability maturity level that the organization is required to successfully perform. For each process area, objectives, specific practices and procedures are defined.

CMM implementation experience

At this point it is worthwhile to quickly review some success stories reported by companies that achieved level 5 assessment according to CMM, the efforts invested and benefits gained. For two of the companies, ISO 9000 certification represented a preparatory step for their final goal of achieving CMM level 5 assessment. In addition, we relay some of the experience accumulated with CMM implementation by a consulting firm.
The following cases are presented:
■ Boeing's Space Transportation Systems Software
■ Tata Consultancy Services (TCS)
■ Telcordia Technologies
■ Gartner Inc.

Boeing's Space Transportation Systems Software

Wigle and Yamamura (1999) discuss the three-year process of gradual quality assurance improvements that finally yielded the CMM level 5 for Boeing. The improvements realized in level 5 projects included:
■ A substantial shift in defect detection, from 89% late detection by testing to 83% early detection by application of various review methods.
■ Earlier detection of defects caused a 31% decrease of rework efforts.
■ Elimination of defects prior to version release increased from 94% to almost 100%.
■ A 140% increase in general productivity.

Tata Consultancy Services (TCS)

TCS's quality project, summarized by Keeni (2000), was implemented by a South Asian company employing a staff of 14 000. For TCS the CMM project was a natural continuation of its successful adoption of ISO 9000 standards. It required two years (1992–1994) for the company to adapt its procedures and entire quality management system (QMS), which culminated in ISO certification of all the company's major centers. After the new QMS was firmly established, the company continued on to the CMM project in 1996. As TCS was ISO 9000 certified, very few practices needed adaptation to achieve the CMM level 3 assessment. The next phase

involved a pilot project, initiated in one of TCS's centers (professional staff of 1000). The pilot project's goal was to achieve level 4 assessment for the center, which was achieved in 1998. Similar projects were launched in 1997 in two additional centers; their target – level 5 assessment – was achieved in 1999. In 1998, TCS decided to expand its CMM project to embrace 17 of its development centers in India. By 2000, a significant proportion of those centers had achieved level 5 assessment, while others had reached level 4 assessment. One of the company's major efforts was certification of the software quality assurance professionals who were to lead the ISO 9000 and CMM quality projects. The SQA professionals certified included:

- Three authorized CMM lead assessors
- Some 77 internally trained CMM assessors
- Some 678 certified quality analysts
- Over 300 quality auditors.

Among the main benefits listed by the company are the following improvements, achieved during 1996–2000:

(1) Reduction of average percentage of rework from 12% to about 4%
(2) Reduction of percentage of project schedule slippage from over 3% to less than 2.5%
(3) Increase in overall review effectiveness from 40% to 80% defect detection
(4) Decreases of 5% in management efforts and of 24% in change request implementation efforts.

Telcordia Technologies

Telcordia Technologies traveled a remarkable journey from low quality software development to ISO 9000 certification and CMM level 5 assessment, as analyzed by Pitterman (2000). Development efforts by the company's software quality assurance system and progress control teams (about 2% of overall software development staff) began in 1994. As its goals, the software quality assurance team set ISO 9000 certification as its primary objective, followed by CMM assessment. All company software development units were ISO 9000 certified by September 1996.

The now well-established quality system required relatively limited additional efforts to achieve CMM level 3 assessment, achieved in December 1996. However, the next stage, CMM level 5 assessment, required substantial efforts for development of quantitative quality assurance tools and further development of the QMS. In May 1999, eight development units, employing more than 3500 software engineers, had successfully realized this goal. Among the main benefits garnered by Telcordia during its six-year quality journey, we can cite:

- A 94% reduction in the field faults (release faults) density
- Percentage of on-schedule major releases reached 98%
- Overall customer satisfaction rose from 60% in 1962 and 80% in 1994 to over 95% in 1997.

Gartner Inc.

Gartner Inc. is a consulting firm that specializes in CMM implementation. A report (Gartner Inc., 2001) summarizing the firm's accumulated experience presents some quantitative data to support its claims. Of special interest are the results dealing with the benefits of CMM application and the time required for progress from one capability level to the next. The data should, however, be treated with some reservations as the total number of organizations observed and the period over which the data were collected are not mentioned.