Chapter 6: Commercial Best Practices Standards

No single standard is available in the computer industry. Most system developers design their own standards and follow them in their organizations. These standards are variations of the basic IT system development goals and principles. This chapter provides a study of the current commercial best practices standards and tailoring standards guidelines.

Commercial System Development Standards

Commercial system development standards originate from the Institute of Electrical and Electronics Engineers (IEEE) of Europe and Canada and the National Aeronautics and Space Administration (NASA). Almost every sizable computer organization has its own established standard for system development. These standards outline the system development life cycle and the content of required documents. Standards also define system software quality assurance (QA), configuration management (CM), requirement traceability, system design, implementation, testing, and independent verification and validation because they are needed for embedded systems.

System development standards are necessary so that a computer system can interoperate among others. Standards establish uniform system engineering techniques that are applicable throughout the system life cycle. These standards incorporate the best practices that will be cost−effective from a system life−cycle perspective. Standards are intended to be dynamic and responsive to the rapidly evolving system development phases. Data item descriptions (DIDs) that are applicable to the standards are available and provide a set of complete and concise documents that record and communicate information generated from specified requirements.

Characteristics of a Good System Development Standard

A good system development standard is tightly composed and concentrates on system development goals and principles. The standard establishes uniform requirements for system development that are applicable throughout the system life cycle. The requirements of the standard provide the basis for insight into system development, testing, and customer evaluation efforts. Some characteristics of a good standard are listed in Box 6−1.

Box 6−1: Characteristics of a Good System Development Standard

---

- Has a proper structure that represents clear thinking and a standardized means of communication
- Establishes uniformity
- Follows system development discipline
- Fits in accordance with system development goals and principles and can be used with other standards
- Follows the processes, standards, and procedures (rigorous)
- Provides guidelines, references, road maps, and checkpoints
- Involves quantitative feedback reviews and audits
- Is user friendly and not user hostile
- Includes reasonable documentation
- Provides quality activities and evaluations
- Provides visibility into system development status
- Provides a template model that will introduce a suitable system development method and tools

- Encourages use of reusable components
- Encourages tailoring aspects
- Establishes tests for the system environment
- Standardizes system development and management processes
- Provides means that will help develop systems that are efficient and cost−effective throughout a product's life cycle.

A good standard defines the requirements for an enterprise's total technical effort related to development of products that include hardware and software and processes that provide life−cycle support for the products. The objectives are to provide high−quality products and services with the desired practitioners and performance features at an affordable price and on time. This involves developing, producing, testing, and supporting an integrated set of products that includes hardware, software, networking, practitioners, data, facilities, and material. The processes include services and techniques that are acceptable to customers, stakeholders, and users.

System Management Standard

The system management standard prescribes an integrated technical approach to engineering of an IT system and requires the application and management of the system's engineering process.

IEEE Standards

IEEE standards provide recommendations that reflect the state−of−the−art system engineering principles for development and maintenance. The IEEE standards meet the requirements of a costly, complex systems' quality, performance, and reliability. The purpose of a standard is to establish and apply requirements during the acquisition, development, and support of software systems. IEEE has designed the following standards for system development:

- P730, Software quality assurance plans
- P828, Standard for software CM
- P829, Standard for software test documentation
- P830, Guide for software requirements specifications
- P1012, Standard for software verification and validation
- P1016, Recommended practices for software design description
- P1058, Standard for software project management plans
- P1061, Standard for software quality metrics methodology
- P1062, Practice for software acquisition
- P1219, Standard for software maintenance
- P1220, Standard for application and management of the systems engineering process
- ISO/IEC 12207, Standard for information technology−software life−cycle processes
- P1233, Guide for system requirements specifications

- P1362, Guide for information technology−system definition−concept of operations document
- P14143, Information technology−software measurement
- P1420, Software reuse
- P1471, Recommended practice for architecture description

Standard for Application and Management of the Systems Engineering Process

The IEEE P1220 standard is for application and management of the systems engineering process. The purpose is to provide a standard for managing a system from initial concept through development, operations, and disposal. This standard defines the interdisciplinary tasks that are required throughout a system's life cycle to transform customer needs, requirements, and constraints into a system solution. The standard guides the development of systems that include humans, computers, and software for commercial and industrial applications. This standard applies to an enterprise within an enterprise that is responsible for developing a product design and establishing the life−cycle infrastructure needed to provide for life−cycle sustainment.

This standard specifies the requirements for the system's engineering process and its application throughout the product's life cycle. The standard does not define the implementation of each system life−cycle process but addresses the issues associated with defining and establishing supportive life−cycle processes early and continuously throughout product development. In addition, the standard does not address the many cultural or quality variables that must be considered for successful product development. The standard focuses on the engineering activities necessary to guide product development while ensuring that the product is properly designed to make it affordable to produce, own, operate, maintain, and eventually dispose of without undue risk to health or the environment.

The P1220 standard describes an integrated approach to product development that represents the total technical effort for the following:

- Understanding the environments and the related conditions in which the product will be used and for which the product must be designed to accommodate
- Defining product requirement in terms of functional and performance requirements, quality factors, usability, producibility, supportability, safety, and environmental influences
- Defining the life−cycle processes for manufacturing, testing, distribution, support, training, and disposal, which are necessary to provide support for products

The standard covers the system engineering management plan, general requirements, the system engineering process, and application of the systems engineering throughout the system life cycle and a list of documentation that will be delivered for maintenance. The general requirements include the following systems engineering processes:

- Requirements analysis and validation
- Functional analysis and verification
- Synthesis
- Design verification
- Systems analysis
- Control

The general requirements also include the following:

- Policies and procedures for systems engineering
- Planning of the technical effort
- Development strategies
- Modeling and prototyping
- Integrated database
- Product and process data package

- Specification tree
- Drawing tree
- System breakdown structure (SBS)
- Integration of the systems engineering effort
- Technical reviews
- Quality management
- Product and process improvement

The application of systems engineering throughout the system life cycle includes the following:

- System definition stage
- Preliminary design stage
- Detailed design stage
- Fabrication, assembly, integration, and test stage
- Production and customer support stage
- Simultaneous engineering of products and services of life cycle processes

Standard for IT: Software Life−Cycle Processes

Software is an integral part of IT. The 12207 standard is for the software life cycle. The standard is produced by the International Standards Organization (ISO) and the International Electro−Technical Commission (IEC). The Electronic Industries Association (EIA) also contributed in this standard. This standard provides industry with a basis for software practices. The ISO/IEC 12207 standard is packaged in the following three parts:

- IEEE/EIA 12207.0 is the standard for information technology−software life−cycle processes. This standard contains basic concepts, compliance, life−cycle process objectives, and life−cycle data objectives.
- IEEE/EIA P12207.1 is the guide for ISO/IEC 12207 and called the *standard for information technology*−software life−cycle processes−life−cycle data. This standard provides additional guidance on recording life−cycle data.
- IEEE/EIA P12207.2 is the guide for ISO/IEC 12207 and is called *standard for information technology*−software life−cycle processes−Implementation considerations. This standard provides additions, alternatives, and clarifications to the ISO/IEC 12207's life−cycle processes.

A proliferation of standards, procedures, methods, tools, and environments are available for development and management of software. This proliferation has created difficulties in software management and engineering, especially in integrating products and services. The software discipline needs to migrate from this proliferation to a common framework that can be used by software practitioners to speak the same language to create and manage software. The 12207 standard provides such a common framework. This framework covers the life of software from conceptualization of ideas through retirement and consists of processes for acquiring and supplying software products and services. In addition, the framework provides for controlling and improving these processes.

The highlights of this standard are as follows:

- It is open ended rather than the specification of any particular software development method.
- The software developer is responsible for selecting software development methods and CASE tools that best support the customer's requirements.
- It provides the means for establishing, evaluating, and maintaining quality in software and associated

documents.

- The IEEE standard is further modified so that it will be acceptable as an international standard and be a part of the ISO 9000 standard.
- Major features eliminate waterfall, top−down, and hierarchic implications.
- It eliminates the functional area tracks (e.g., software development management, software engineering, etc.), which tend to be waterfall oriented.
- It explains several life−cycle models, tells how an incremental or evolutionary development is planned, and provides guidance for the selection of appropriate deliverables in each increment.
- It eliminates the requirements that partition the computer software configuration item (CSCI) into computer software components (CSCs) and computer software units (CSUs). This requires only a CSCI to be partitioned into components and uses the method that is proposed in the software development plan (SDP) that the software developers create.
- It requires that the software developer lay out a software development process and that it conforms to the life−cycle model, which has been established for the software.
- It acknowledges that each software development activity is an update or refinement of what has gone before rather than a first−time occurrence.
- It reorganizes activities that occur at the same time into activities that are concerned with a given software development activity.
- It explains how each activity is interpreted in the context of builds (e.g., How should the planning activity be divided among builds? How should requirements analysis, design, coding, testing, and product evaluation be performed in incremental builds?).
- It covers in−process reviews and supplements or substitutes for formal reviews.
- It identifies system requirements analysis and system design separately.
- It eliminates the document−driven implications of the ISO 9000 standard.
- The language of the standard is changed from 'software and documentation' to the more generic 'software development products,' which acknowledges nondocument representations, such as data in CASE tools.
- It separates activity requirements from documentation requirements and emphasizes that software development activities need not result in documents.
- It clarifies that deliverable documents are the required outcome of an activity only when so specified on a contract data requirements list (CDRL).
- It is explicit in acknowledging electronic representation of information in lieu of documents.
- It adds guidance about ordering the executable (and possible source) code via contract line item number (CLIN) rather than on a CDRL.
- It adds explicit permission and delivers CASE tool contents for CDRL.
- It clarifies and makes the customer's requirements more consistent for each level of testing.
- It provides clear guidelines for software support.
- It expands and clarifies the requirements, which concern incorporation of reusable software.
- It specifies that nondevelopmental items (possibly modified) will be incorporated into systems under development if they meet user needs and will be cost−effective over the life of the system.
- It interprets this policy for software by itemizing considerations that determine if a reusable software component will meet user needs and be cost−effective over the life of the system.
- It requires that the software developer analyze candidate reusable components in light of these considerations, report the findings and recommendations to the customer, and incorporate reusable components that meet the criteria.
- It specifies allowable substitutions for this standard's required documentation when reusable software is incorporated.
- It makes a preliminary statement about other allowable substitutions (e.g., in testing and formal reviews).
- It defines clearly the interface between software testing and system testing.
- It has a section for planned software QA requirements.

- It adds a requirement that will identify, collect, and apply management indicators, and it provides a list of candidate indicators that aid in the selection of a set.
- It revises the requirement on risk management with emphasis on risk as a guiding principle for planning and managing projects.

Box 6−2 contains a list of 22 individual DIDs that are applicable to this standard.

Box 6−2: List of Data Item Descriptions

Plans

- Software development plan (SDP)
- Software installation plan (SIP)
- Software support plan (SSP)

Concept and requirements

- Operational concept document (OCD)
- System/segment specification (SSS)
- Software requirements specification (SRS)
- Interface requirements specification (IRS)

Design

- System/segment design document (SSDD)
- Software design document (SDD)
- Interface design document (IDD)
- Database design document (DBDD)

Test

- Software test plan (STP)
- Software test description (STD)
- Software test report (STR)

User or operator

- Software user's manual (SUM)
- Software input/output manual (SIOM)
- Computer center software operational manual (CCSOM)
- Computer system operator manual (CSOM)

Support

- Version description document (VDD)
- Software product specification (SPS)
- Firmware support manual (FSM)
- Computer instruction set architecture

A set of six consolidated DIDs combines the DIDs for plans, requirements, design, testing, user or operator manuals, and support; a single DID for small projects summarizes all other DIDs. These DIDs describe a set of documents that record the required information by this standard. The manager should produce deliverable data that use automated techniques.

ISO 9000 Standards

ISO 9000 standards are published by the ISO. The ISO 9000 series consist of the following quality standards:

- ISO 9000
- ISO 9001
- ISO 9002
- ISO 9003
- ISO 9004

ISO 9000 is an overview for selecting the appropriate standard. ISO 9001 covers the 20 elements of an effective quality management system (QMS), which include design, production, servicing, and installation:

1. Management responsibility
2. Quality system
3. Contract review
4. Design control
5. Document and data control
6. Purchasing
7. Control of customer−supplied product
8. Product identification and traceability
9. Process control
10. Inspection and testing
11. Control of inspection measuring and test equipment
12. Inspection and test status
13. Control of a nonconforming product
14. Corrective and preventive action
15. Handling, storage, packaging, preservation, and delivery
16. Control of quality records
17. Internal quality audits
18. Training
19. Servicing
20. Statistical techniques

ISO 9002 addresses all elements of a QMS except design. ISO 9003 focuses on final inspection and testing. ISO 9004 provides implementation guidance and can be used to expand and improve an organization's quality system. If ISO 9004 is in place, an organization can achieve the ISO 9001 standard.

The ISO 9000 series was first published in 1987 and is currently under review. Under the revision, ISO 9002 and ISO 9003 will be incorporated into a new ISO 9001. The new version of ISO 9001 will include the following:

- All types of products and services
- A single standard with flexibility of use
- A more generic approach to conform to different types of industries

- Alignment with other standards
- Consistent terms, phrases, and definitions

According to James S. Bigelow, a member of the ISO Technical Committee, several boundaries have been identified for the revised ISO 9001:

- Preventing customer dissatisfaction rather than achieving a competitive advantage
- Complying with requirements instead of offering guidance
- Seeking effectiveness over efficiency
- Seeking minimum QA requirements rather than best practices
- Meeting customer requirements rather than enhanced expectations
- Seeking pass/fail instead of degree of performance
- Focusing on improving processes by reducing risks and preventing failures

ISO 9001 is an independent standard that can adapt within reason to meet the needs of an organization (Fellenstein, 1999).

Tailoring Standards Techniques

Tailoring standards for an individual application is essential. Tailoring means the selection of only those products, activities, and reviews that fit the characteristics and are essential to a particular project. The purpose of tailoring helps in the evaluation of requirements in a standard that will save money, prevent duplication, and preserve the schedule of the project. The system developer can recommend tailoring the standards, but ultimately the customer, users, and stakeholders make the final decision.

The result of the tailoring process is reflected in the statement of work (SOW) that prescribes the tasks and reviews. The CDRL contains all of the deliverable documentation. Sometimes clarification is necessary rather than a deletion of requirements. In these cases, the DID for a product can be modified, which will make it unique to the project. When tasks are being added, they may be included in the SOW, which leaves the DID unchanged.

The following are factors that the manager should consider when tailoring a standard:

- System development process that will be used
- System characteristics and intended end use
- Acquisition strategy and type of project management
- Acceptable risk
- Schedule
- Budget
- Development visibility required
- System maintenance concept

Tailoring standards also depends on the class of software. Software classification includes operational, support, system, diagnostic, and automatic test equipment. The tailoring factor changes for the software that is being developed, modified, and reused. Commercial off−the−shelf (COTS) software or nondevelopmental items (NDI) also affect the tailoring factor.

Guidelines for Tailoring

The tailoring process is performed many times throughout the life cycle of a system. Many different events trigger the need for the tailoring of the standards and DID. Tailoring should take place every time the system enters a new phase in the life cycle. Phases include concept exploration, demonstration and validation, full−scale development, and production and deployment. The levels and types of documentation that are needed for each phase vary and should be reflected in the tailoring process. The suggested guidelines for tailoring a standard are as follows:

- Classify the required system by development or nondevelopment category.
- Select activities and reviews in accordance with applicable standards and DID.
- Select deliverable products.
- Tailor the DID.

IT Project Standards Checklist

- Establish project management standards that include the following:

  Project progress reporting process

  Statement of work−clarification procedures

  Project management graphic representations

  Staff qualifications and experiences

  Cost analysis

  Schedule influence

  Risk analysis management process

  Metrics indicators

  Review process and procedure

  Tailoring standards plan

  Documents deliverable process

  Education and training plan

  Effective dialog and communication process between system developers and customers, users, and stakeholders

  System delivery process

  System products acceptance schema
- Establish a system requirements analysis standard that includes the following:

System requirements analysis graphic diagrams

Maintenance of a system requirements list

System external interfaces

Modeling of customer requirements

Requirements feasibility analysis

Requirements traceability schema

Determination of system complexity

System hardware diagram

- Establish a system design standard that includes the following:

System design graphic representations

Allocation of software and hardware requirements

Software development plan

Selection of a suitable method

Selection of a suitable computer−aided software engineering (CASE) tool

Planning for software quality

Planning for increment software development

Planning for software testing

Planning for CM

Planning for corrective action

Reviewing and auditing by the customer

System behavioral design

System architectural design

- Establish a software requirements analysis standard that covers the following:

Software requirements analysis graphic diagrams

Maintenance of software requirements list

Creation of prototyping models

Interface requirements

Establishment of a database

Establishment of a real−time influence

Determination of sizing and timing requirements

Software requirements traceability

Plan for testing requirements

- Establish a software design standard that contains the following:

Architecture design diagrams

Behavioral design diagrams

Object diagrams

Determination of algorithms

Data structure

Data types

Associated operations

Database logical design

Formation of packages, subprograms, functions, and tasks

Functional cohesion

Data coupling

Software design requirements traceability

Compilation dependencies diagrams

Identification, raising, and handling of exceptions

Setting up of a software design file

Software reuse schema

Interfaces

Plan for testing design

- Establish a code style standard that includes the following:

  - ♦ Coding formatting

  Horizontal spacing

Indentation

Alignment of operators

Alignment of declarations

Blank lines

Paginations

Number of statements per line

Source code line length

- Readability

Underscores

Numbers

Capitalization

Abbreviations

- Commentary

General comments

File headers

Unit function description

Marker comments

Highlighting

- Naming conventions

Names

Type identification

Object identification

Program unit identification

Constants and named numbers

- Using types

Declaring types

Enumeration types

Overloading enumeration littorals

- High−level program structure

  Separate compilation capabilities

  Subprograms

  Functions

  Packages

  Functional cohesion

  Data coupling
- Syntax

  Loop names

  Block names

  Exit statements

  Naming and statements
- Parameters lists

  Formal parameters

  Named association

  Default parameters

  Mode indication
- Types

  Derived types

  Subtypes

  Anonymous types

  Private types

  Data structures

  Heterogeneous data

  Nested records

  Dynamics data
- Expressions

  Range values

Array attributes

Parenthesized expressions

Positive forms of logic

Short−circuit forms of the logical operators

Type−qualified expressions and type conversion

Accuracy of operations with real operands

♦ Statements

Nesting

Slices

Case statements

Loops

Exit statements

Safe statements

'Go to' statements

Return statements

Blocks

♦ Visibility

Use clause

Rename clause

Overloaded subprograms

Overload operators

♦ Exceptions

Disasters versus state information

User−defined, implementation−defined, and predefined exceptions

Handlers for others

Propagation

Localization of the cause of an exception

♦ Erroneous execution

Unchecked conversion

Unchecked deallocation

Dependence on parameters−passing mechanism

Multiple address clauses

Suppression of exception check

Initialization

♦ Tasking

Tasks

Task types

Dynamic tasks

Priorities

Delay statements

♦ Communication

Defensive task communication

Attributes count, collable, and terminated

Shared variables

Tentative rendezvous constructs

Communication complexity

♦ Termination

Normal termination

Abort statement

Programmed termination

Abnormal termination