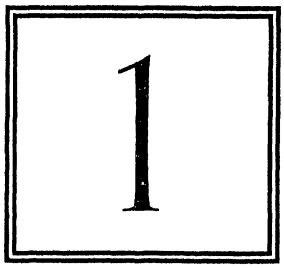
INTRODUCTION



On an outside wall of the thirteenth century Town Hall in Rothenburg-ob-der-Tauber, Germany hangs an equally old iron *standard* used to measure the length of loaves of bread. Subject to punishment were bakers who made their loaves too short—for cheating their customers—as were those who made their loaves too long—for raising unrealistic expectations. The goal of this chapter is to provide general material about software engineering and software engineering standards and to address the reader's expectations regarding the usefulness of the standards.

Software Engineering

The Institute of Electrical and Electronics Engineers defines *software engineering,* in IEEE Std 610.12, as:

> *(1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, that is, the application of engineering to software. (2) The study of approaches as in (1).*

Most of the standards to be considered are *practice* standards rather than *product* standards, concerned with the regulation of the practice of software engineering rather than the interfaces of the products produced.

Is it Engineering?

One could be excused for denying the premise of this book. Software engineering is not among the 36 engineering professions licensed in the United States. Furthermore, 48 states have laws forbidding an unlicensed individual from advertising as an "engineer." The state of Texas has prohibited its universities from offering master's degrees in software engineering and the state of New Jersey has considered legislation requiring the licensing of all software professionals [Jones95].

Nevertheless, in deference to common usage, this book will use the term "software engineering"; readers may choose to view the term as a statement of a goal or ideal rather than as a statement of a fact.

Engineering can be viewed as a closed feedback loop as shown in Figure 1. An engineering *process* consists of related activities performed in response to a statement of *needs* and consuming *resources* to produce a *product*. In order to manage or improve the process, one must exert *control*. Control is a decision-making mechanism that considers *goals* and *constraints* in the formulation of *action* that is intended to direct or modify the process. The decision to take action is based on *measurements*, quantitative evidence

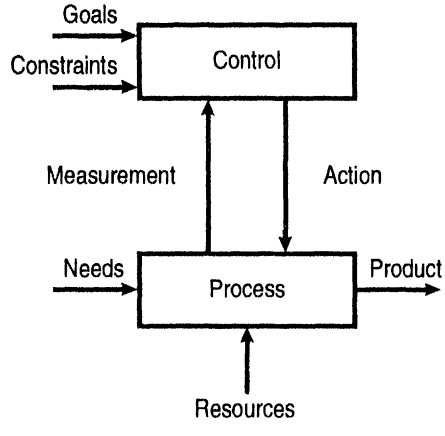


Figure 1. A model of engineering. (Source: [SESC93].)

regarding the state of the process. Measurements can be made of conditions inside the process, products of the process, and the satisfaction of users of the products [SESC93]. We would expect software engineering standards to contribute to the implementation of such a model with respect to the development, operation, and maintenance of software systems.

Indeed, we will find standards related to the process, product, and resources involved in the software discipline. We will find standards describing the treatment of needs in software development, not only as requirements and specifications, but articulating less formal needs from more remote stakeholders. We will find standards describing management plans, measurements, and ac-

tions for the purpose of controlling the ongoing processes. Finally, we will find methods to articulate goals and constraints, even informal ones, to guide the managers.

So, even if the software discipline has not yet formalized the empirical underpinnings of an engineering discipline, we will find that, in many ways, it is acting as if it has.

Relationship to Other Disciplines

Software engineering occupies a position intermediate between, on the one hand, the mathematical and physical disciplines *of computer science and technology* and, on the other hand, the requirements of the particular *application domains* applying the findings of the former to solve problems of the latter (Figure 2). The techniques for the engineering of software can be viewed, in part, as specializations of those of more general disciplines, such as *project management, system engineering,* and *quality management.* Furthermore, a software project must implement requirements imposed by cross-cutting disciplines like *dependability* (a term more general than *reliability*) and *safety*. These contextual disciplines are

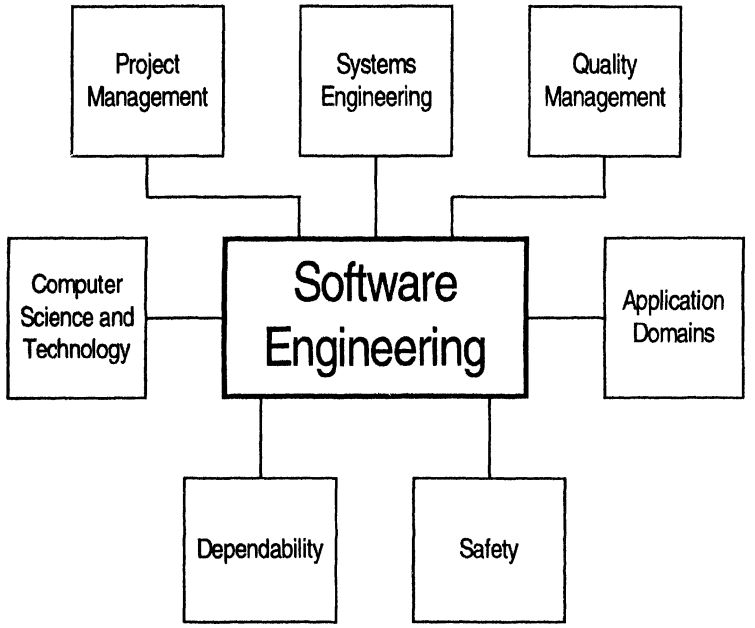


Figure 2. Relationship of software engineering to other disciplines.

important to the book because subsequent chapters will use them as entry points for selecting appropriate software engineering standards.

Fundamental Principles

The engineering style of education deals with rapidly changing technology by teaching fundamentals; students are provided little choice regarding their curriculum. As a consequence, there is a set of things that we can expect every engineer to know [Parnas95]. The teaching material is based on a common, principle-based body of knowledge, codified by some more-or-less officially designated organization, often enforced by licensing requirements.

Relating to the engineering style of education, Tom Gilb [Gilb96] offers a definition of engineering that he credits to Billy Koen:

> *Engineering is the use of* principles *to find designs that will meet multiple competing objectives, within limited resources and other constraints, under conditions of uncertainty.*

In other fields, practice standards can be traced to a body of fundamental scientific and engineering *principles* that constrain the standards. A trivial example is that mechanical engineering standards are constrained by the three-dimensional geometry of physical objects. The codification of software engineering standards is faced by particular challenges in this area. First, the subject of the standards—software—is inherently intangible and unconstrained by the common laws of physics. Second, the discipline is relatively new, compared to other engineering disciplines, and many of its important concepts remain immature. Third, there is not yet a commonly accepted body of knowledge[1] that can serve as a foundation, nor any body empowered to develop it [Abran96]. Finally, unlike product interface standards, there are few market forces to cause convergence on selected technologies.

This has caused some problems. Unfettered by any integrating forces of principles or dominant products, most software engineering standards are ad hoc recordings of individual practices claimed to be "best." This is not bad when each standard is considered in isolation, but when the standards are considered as a body, we find them to be *dis*-integrated, capriciously different in detail, overlap-

[1]There have been some attempts, though; [Davis95] is a catalog of principles.

ping and occasionally contradictory. These characteristics put the erstwhile adopter of the corpus of standards into a difficult situation, because the adopters must themselves develop some mechanism to rationalize, explain, and relate the various standards chosen for implementation.

Part of the solution to the problem is the adoption of a framework of vocabulary, key relationships and other constraints to which each individual standard must adhere. In fact, IEEE Software Engineering Standards Committee (SESC) and its international counterpart, ISO/IEC JTC1/SC7, have taken steps in this direction, efforts that are described elsewhere in this book. Perhaps a more basic requirement is for the identification of a set of *fundamental principles* that would serve to explain and motivate the provisions of the various standards. IEEE SESC is initiating this step.

Figure 3 shows a notional depiction of the role of principle standards. The principles of software engineering would be regarded as selected, adapted and specialized from the principles of engineering in general. The provisions of practice standards would be motivated by the software engineering principles and would be traceable to those principles. So-called "best practices" would be viewed as the detailed implementation of the provisions of the practice standards. Viewed in the other direction, practice standards would be regarded as descriptions of observed, effective best practices and the principles as abstractions of the practice standards. The principles found to be relevant beyond the scope of software engineering (perhaps those related to complexity, for example) might be considered as general principles of engineering.

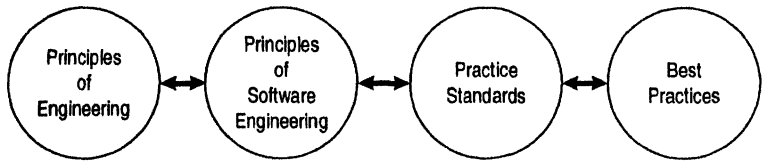


Figure 3. Relationship of principles and practice.

In this model, we can view practice standards as existing in a tension between the consolidating and integrating forces exerted by the principles and the expansive and innovative forces exerted by the recognition of new, effective practices.

IEEE SESC has initiated an effort to identify fundamental principles for software engineering. A workshop at the 1996 Software Engineering Standards Forum considered candidate princi-

ples and developed a set of criteria to be applied to candidates [Jabir97]. A Delphi experiment involving a number of notable experts in software engineering identified an initial working set of principles during 1997. A follow-up workshop held at the 1997 International Software Engineering Standards Symposium reviewed the results of the Delphi experiment and produced a working set of principles. More refinement is planned.

Software Engineering Standards

For the purposes of this book, the characterization of standards provided in [SESC93] is helpful:

> *A standard can be: (1) an object or measure of comparison that defines or represents the magnitude of a unit; (2) a characterization that establishes allowable tolerances or constraints for categories of items; and (3) a degree or level of required excellence or attainment. Standards are definitional in nature, established either to further understanding and interaction, or to acknowledge observed (or desired norms) of exhibited characteristics or behavior.*

Most of the standards described in this book have been developed by Standards Developing Organizations (SDO) operating on the principles of consensus development and voluntary adoption. For the purposes of this book, the organizations are regarded as national (US, with a few exceptions) or international. It must be noted, however, that even this basic distinction over-simplifies the actual situation.[2] Many major standards developers, for example, the IEEE, include members from many nations, any of whom may contribute to the development of standards. The resulting standards may be adopted by anyone, anywhere on the globe. Nevertheless, for purposes of international standardization, these *trans-national* organizations are regarded as national entities and required to participate via the designated national body. For example, IEEE contributions to international standardization are administered by the American National Standards Institute (ANSI)

---

[2]It turns out that almost every general statement made about standardization is an oversimplification. It seems that nearly every organization and every project involves special circumstances that make it exceptional in some regard. The term *standards organization* may be a not-so-funny oxymoron.

despite the fact that the standards were developed by members from many nations.

Some notable characteristics of standards developed by US organizations are listed in [Cargill97]:

- They have been written by a committee of anyone who could attend the meetings.
- They have undergone public scrutiny.
- All technical comments have received responses.
- They are a product of consensus within the committee and within an industry segment or professional community.

Subject to procedures administered by the American National Standards Institute (ANSI), such a standard may be designated as an "American National Standard." Policies of the US government[3] provide that these national standards may be used in federal procurements [Cargill97].

Scope of Software Engineering Standards

Software engineering standards cover a remarkable variety of topics. ISO/IEC DTR 14399 (Version 3.0) lists 21 subject areas. For the purpose of organizing their book [Magee97], Stan Magee and Leonard Tripp organized 29 subjects into the three categories adapted for Table 1.

Table 1. Scope of Software Engineering Standardization

Process	Technique or Tool	Applicability
Acquisition	CASE tools	General
Requirements definition	Languages & notations	Defense
Design	Metrics	Financial
Code and test	Privacy	Medical
Integration	Process improvement	Nuclear
Maintenance & operations	Reliability	Process control
Configuration management	Safety	Scientific
Documentation	Security	Shrink-wrap
Project management	Software reuse	Transportation
Quality assurance	Vocabulary	
Verification and validation		

[3]OMB Circular A119.

Importance of Software Engineering Standards

To consider the importance of software engineering standards, one must consider the uses to which they are put and the benefits that accrue from their application.

Improving the Product

Nearly all of the standards discussed in this book are voluntary, that is, an organization makes its own decision, without coercion, to adopt the standard. (This contrasts with regulatory standards, imposed by processes similar to law, and mandated standards, such as military standards, required as a precondition of doing business with a dominant customer.) Organizations often adopt these standards because they improve their products, or improve the perception of their products in a competitive marketplace. Alternatively, the standards may improve the organization's business processes, allowing them to make their products more cost-effectively.

Examples of benefits that standards may provide in this regard are the following:

- Some standards are simply statements regarding subjects in which the uniformity provided by agreement is more important than the gains to be made by small, but local, improvements. Standards on terminology, e.g. IEEE Std 610.12, and notations, e.g. the draft IEEE standards on the IDEF notation, are examples.

- Some standards provide a nomenclature for complex concepts, which, absent standardization, could exhibit detailed differences in characteristics which might ultimately prove crucial. For example, IEEE Std 1028 provides minimum, essential characteristics for the type of review known as an *inspection*.

- Some standards, in the absence of scientific proof of validity, provide criteria for measurement and evaluation techniques that are at least validated by consensus wisdom. For example, IEEE Std 1061 provides a methodology for metrics that can be used as early indicators of later results.

- Some standards record a community consensus of "best practices," that is, techniques broadly regarded as generally effective. For example, IEEE Std 1008 provides re-

quirements and recommendations for the unit testing of code.

- Some standards provide a unified and systematic treatment of the so-called "ilities" in a manner that cuts across the organization of many enterprises, hence effecting consistent internal treatment. An example is IEEE Std 730 on software quality assurance.
- Some standards provide a framework for communication with customers and suppliers, reducing misunderstanding, and shortening the time (and text) needed to reach agreement. An important example is IEEE/EIA Std 12207 on software life cycle processes.
- Some standards share techniques that can lead to qualitative improvements in developing software better, faster, or less expensively, for example, IEEE Std 1044 on software anomaly classification.

To be sure, not all observers agree that software engineering standards have been successful. [Pfleeger94] says that the "standards lack objective assessment criteria, involve more process than product, and are not always based on rigorous experimental results." [Fenton96] finds "no evidence that any of the existing standards are effective [in improving] the quality of the resulting software products cost-effectively." [Schneidewind96], though, points to success stories such as the organization that has produced nearly error-free code for the space shuttle, in part, through use of software engineering standards. All parties would agree, though, that improvement is desirable.

Protecting the Buyer

With many products, buyers can make appropriate decisions based on advertising literature, previous experience with the seller or direct examination. The increasing complexity of technology-based products, however, inevitably causes essential characteristics to remain hidden until after purchase. Standards can play a role when they provide accurate information regarding the suitability of products for specific uses [Brobeck96]. The product quality standards of ISO/IEC JTC1/SC7/WG6 are aimed in this direction. For the most complex of systems, the application of standards by the developer can serve to increase the buyer's confidence in the seller's methods for coping with that complexity. The standards applied by the avionics and nuclear industries are examples.

Protecting the Business

Courts in the United States have used voluntary standards to establish a supplier's "duty of care." Failure to adhere to standards does not necessary establish negligence but may be considered as evidence when dealing with issues like product safety and liability [Peach94, p. 322]. On the other hand, adherence to the appropriate standards is a strong defense that the supplier was not negligent in its development practices and has taken due care to deliver a product that is safe and fit for its intended use. The introduction of evidence that a product meets a voluntary standard is admissible in 47 of the 50 states [Batik92]. Increasingly, companies are developing liability prevention programs that incorporate voluntary standards as key parts.[4] IEEE Std 1228 for software safety plans, might be appropriate for such usage.

Florida Power and Light (a winner of the prestigious Deming Award in 1990 [Batik92]), not only applies software engineering standards but sometimes performs causal analysis back to the standards themselves when failures are noted. A major credit reporting firm has applied the entire corpus of IEEE SESC standards to its organizational software development processes to further bolster its defenses against claims of reckless development, maintenance, and operation of its databases and their accompanying software.

Even in contractual situations, the appropriate application of standards protects both parties by dividing up responsibilities, clarifying terminology, and defining expected practices. Examples of standards appropriate for this purpose include IEEE/EIA 12207 and EIA/IEEE J-Std-016.

Increasing Professional Discipline

Even if the practice of software is not yet a proper engineering discipline, it is moving in that direction. A body of practice standards is an essential step because it would serve to define the

---

[4]Of course, the writers of standards assume some liability for their product—the standard. In a famous case, the Sunshine Mine used, for thermal insulation, a foam material incidentally described as "fireproof" in a standard of the American Society for Testing and Materials (ASTM). A fire at the mine was blamed partly on the poorly written standard and ASTM was named as a co-defendant in the lawsuit. Although the lawsuit eventually was dropped, ASTM and other standards organizations have taken steps to protect themselves, and their voluntary participants [Batik92].

methods to be expected in the responsible practice of software engineering. An example might the software verification and validation (V & V) standard, IEEE Std 1012. A joint task force of the IEEE Computer Society and the ACM is currently investigating other steps necessary to move toward the goal [Jones95].

Introducing Technology

Finally, the Software Engineering Institute notes that standards play a vital role in technology transition. "Standards provide users with common terminology and a framework for communicating about technologies across organizational boundaries. Such communications is particularly critical to acceptance by late adopters." Furthermore, codification of technologies prepares them for adoption [Pollak96]. Examples might include the recommended practices on Computer-Aided Software Engineering (CASE) tool selection and adoption developed by IEEE SESC and now under consideration by ISO/IEC JTC1/SC7.

History

Although *software engineering* may not yet be a recognized branch of engineering, the roots of an organized discipline began to emerge in the 1960s and use of the term itself dates back to the now-famous 1968 North Atlantic Treaty Organization (NATO) conference.

Perhaps because of government's inherent need to conduct its business in a procedural manner, early US software engineering standards were written by organizations within the federal government. In 1973, a task force of the National Bureau of Standards concluded that such standards were feasible. Accordingly, three years later, Federal Information Processing Standard Publication (FIPS Pub.) 38, *Guidelines for Documentation of Computer Programs and Automated Systems*, was published. It was organized around the production of 10 documents: functional requirements; data requirements; system/subsystem specifications; program specifications; database specifications; user manual; operations manual; program maintenance manual; test plan; and test analysis report. Meanwhile, in 1974, the US Navy initiated the development of its Mil-Std 1679, Weapons System Software Development, one of the first standards treating the usage, control, and management of embedded computer resources [SESC93].

IEEE activity began in 1976 with the creation of the Software Engineering Standards Subcommittee (SESS). Its first standard, IEEE Std 730, Standard for a Quality Assurance Plan, was ap-

proved for trial use in 1979 and full use 2 years later. Like FIPS Pub. 38, it was (and remains) oriented toward the documentation requirements, only implicitly placing requirements on the underlying processes.

International standardization activities related to software occurred in various technical committees of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC), depending upon the application area. ISO and IEC agreed in 1987 to form Joint Technical Committee 1 (JTC1) to deal with the area of *information technology* (IT). Nevertheless, we will see that important work strongly related to IT continues in committees of ISO and IEC.

Makers of Software Engineering Standards

The need for software engineering standards has been filled by an amount that some would regard as exceeding the requirement. An authoritative source [Magee97] identifies 315 standards created and maintained by 55 different international, national, sector, and professional organizations; Magee is selective—others identify more.

IEEE SESC alone maintains about 50 standards, if we count "parts" individually. Its planned 1998 collection will be packaged in four volumes comprising a total of about 2,300 pages. The chairman of the SESC, Leonard Tripp, has estimated that the typical SESC standard takes 2 to 4 years to develop and costs (in the labor and travel of volunteers) between $2,000 and $10,000 per page [SESC93].

At the international level, the process is even longer and more expensive. One estimate [Spring95] says that ISO standardization typically exceeds 7 years in duration and the attentive reader will find that at least one family of standards described in this book has been in development for 14 years so far.

At the center of software engineering standardization in the United States is the Software Engineering Standards Committee (SESC) of the IEEE Computer Society. Its collection has grown from 1 in 1981 to about 50 by the end of 1997 and continues to grow at the rate of five or so per year. The size and growth of the collection has exposed many stresses and SESC has been taking the initiative to address the problems apparent in the world-wide corpus of process standards.

The counterpart of SESC in the international forum is ISO/IEC JTC1/SC7. It inherited an obsolescent set of mainframe-oriented

practice standards (on subjects like flowcharts and sequential record processing) when it was formed in 1987 but has turned its attention toward more significant contributions, such as its 1995 standard for software life cycle processes, ISO/IEC 12207. SESC participates in SC7 through its membership in the US Technical Advisory Group (TAG) that formulates national positions and selects the delegation for meetings of SC7.

These two organizations are not alone in their work. Other relevant standards have been written by US organizations including the American Institute for Aeronautics and Astronautics (AIAA), the Electronic Industries Association (EIA), and the Power Engineering Society of the IEEE; national organizations like the Canadian Standards Association (CSA) and Standards Australia; and committees of international organizations, like ISO TC176 (quality), IEC TC56 (dependability), and IEC SC45A and 65A (safety). Important contributions have been made by organizations that are not formally accredited to develop standards, including the International Council on Systems Engineering (INCOSE), the Project Management Institute (PMI), and the Reuse Library Interoperability Group (RIG). Even this list is far from complete; [Magee97] lists 55 organizations that have developed relevant standards—and its authors were selective.

Components of the US federal government sometimes write specifications to regulate their own procurement practices. The role of the Department of Defense is well-known, particularly with respect to development process standards, but the National Aeronautics and Space Administration (NASA) and the Federal Aviation Agency (FAA) have also written standards.

The most influential government agency in information technology standardization, though, has been the Computer Systems Laboratory (CSL) of the National Institute of Standards and Technology (NIST), "where much of the technology infrastructure that is necessary to the United States is either created or validated." NIST is a 1988 renaming of the National Bureau of Standards, formed in 1901. The Computer Systems Laboratory specializes in information technology; it was formed in 1966, as a result of the Brooks Act, to help resolve the problems of incompatible computer systems in the federal government, the world's largest purchases of information technology, by making procurement practices more uniform and enlightened. This was accomplished largely through the creation of the famous Federal Information Processing Standards (FIPS) [Cargill97].

The FIPS publications provided important guidance to government managers on how to make open systems procurements. Complementary processor validation programs ensured that suppliers of open systems met their claims regarding adherence to standards. Process standards were not ignored—NIST/CSL played important roles in the development of standards for life cycle process and high-integrity systems. In recent months, though, CSL has redirected its priorities to specifically targeted efforts, like the National Information Infrastructure.

Roles of Software Engineering Standards

Software engineering standards can play a variety of important roles for an organization. Some of those roles are:

- *Naming*: A standard can provide a succinct name for a complicated concept. Particularly when two parties are contracting for a complex item, it is helpful to have a standard specifying the details. For example, is it easier to judge the adequacy of a 20-page explanation of a supplier's verification and validation procedures or a simple claim that they conform to IEEE Std 1012?

- *Best Practices*: Sometimes organizations want to adopt software development practices that are agreed by the community to represent "best of breed." Practice standards describe practices that are consensually agreed to be sound.

- *Badging*: Organizations need a way to assert (in a supportable fashion) that their institutional practices conform to a constellation of best principles and practices. The need is met by "badges," formulated by expert authorities and sometimes independently certified, for example, ISO 9000 and SEI Capability Maturity Model "levels." This usage is so important that we can expect to see new badges in the near future [Moore95].

- *Contractual Agreement*: In a complex information technology procurement, it is helpful and efficient to decouple complex technological issues from the business aspects of the agreement. Standards such as IEEE/EIA 12207 provide this service by setting norms which may be referenced rather than described in a contract.

Organizational Goals for Using Software Engineering Standards

Not all organizations will have the same goals in the adoption of software engineering standards.[5] Some possibilities are listed here along with the needs that software engineering standards may fill in the achievement of the goals.

Improve and Evaluate Software Competency. An organization may desire processes and measures to calibrate its ability to produce software that is competitive in some or all of the following areas:

- *Quality*: Analyze trends in product and process quality for software organizations.
- *Customer Satisfaction*: Measure the extent to which software satisfies the customers' needs.
- *Cycle Time and Productivity*: Track progress toward goals for software cycle time and productivity improvement.
- *Process Maturity*: Assess progress relative to industry software process benchmarks.
- *Technology*: Assess the application of technology within the organization.

Provide Framework and Terminology for Two-Party Agreements. An organization that specializes in buying or selling software services under contract may desire a uniform framework for defining the relationship and respective responsibilities of the acquirer and the supplier for software and systems containing software, a framework that transcends the scope of any particular contract:

- *Acquisition Process*: Provide the essential actions and criteria to be used by an organization in planning and executing an acquisition for software or software-related services.
- *Supply Process*: Provide the essential actions and criteria to be used by an organization in supplying software or software-related services to an acquirer.

---

[5]The material in this section is based on an untitled point paper written by Leonard Tripp.

- *Life Cycle Processes*: Provide the process requirements to be met during the life cycle (definition, development, deployment, operation, etc.) of software or systems containing software.
- *Life Cycle Deliverables*: Provide the requirements for information to be passed between the supplier and the acquirer during the performance of software life cycle processes.

Evaluate Products of Software Engineering Activities. An organization may need to formulate criteria, processes, and measures to determine the adequacy of a software product to fulfill its mission.

- *External Measurements*: Measurements of completed software products to evaluate the achievement of development goals.
- *Internal Measurements*: Measurement of incomplete software artifacts and development processes to provide early indicators of the achievement of development goals.

Assure High Integrity Levels for Software. An organization may need to develop software for critical applications where safety and dependability are important to protect lives or property.

- *Planning*: A framework to determine that appropriate resources and appropriate controls are provided to ensure treatment of concerns of criticality.
- *Achievement*: Provisions for ensuring that critical requirements for safety and dependability are appropriately treated throughout the providing of the software service
- *Assessment*: Verifiable measurement of the extent to which criticality goals have been achieved.

Trends

Some trends for the future of software engineering standardization are becoming apparent, particularly in the collections of the major organizations involved in developing these standards.

The absence of a firm, empirical and scientific foundation for software engineering standards has been a continuing vulnerability, one that has not gone unnoticed by its critics, for example, [Fenton96] and [Pfleeger94]. With the continuing maturation of the

field, attempts are being made to address this problem. Notable efforts include:

- The series of workshops and other projects, mentioned above, to develop fundamental principles.
- Standardization efforts cooperative with related but more mature disciplines such as systems engineering and project management.
- The so-called "SPICE trials" (more properly termed as the Software Process Improvement and dEtermination project), efforts to empirically validate the process assessment mechanisms of the planned ISO/IEC 15504 series of standards [Emam95].

Little progress toward a coherent discipline can be made when each standard is an individual island of practice unrelated to its peers. Recent years, though, have seen a trend toward the recognition of key standards that provide a framework which may be elaborated by other, more detailed ones. Examples include:

- The broadly recognized quality management framework of the ISO 9000 standards.
- The life cycle process framework of the ISO/IEC 12207 standard on software life cycle processes.
- Cooperative liaison efforts among standards committees concerned with cross-cutting areas like functional safety, dependability, quality, and software engineering.

No one should be surprised at disrespect of software engineering standards if the various collections do not respect and build upon the contributions of other collections. Recent years have seen huge steps toward the harmonization of the important collections. For example:

- SESC standards have been used as the basis upon which SC7 standards have been drafted. On the other hand, SESC has voted to adopt newer SC7 standards to replace their own standards with a similar scope.
- ISO TC176 and IEC TC56 share a single standard, under two numbers, specifying the relationship between quality management and dependability.
- SESC has adopted policies designating various international standards, such as ISO/IEC 12207 and the ISO 9000

series as key standards with which its own must harmonize.

- A major accomplishment of the US adaptation of the ISO/IEC 12207 standard has been the addition of an annex explaining how SESC standards may be used to accomplish the requirements of the 12207 standard. SESC plans a "block change" of those standards during the next 2 years to increase the precision of the fit.

The great success of the SEI Capability Maturity Model and the ISO 9000 quality management standards provide ample indication of the need for "badges" summarizing achievement of important capabilities. It is appropriate for executive managers to deal with badges in order to set corporate level objectives and allocate resources while properly delegating technical activity. Before the end of the millennium, we can expect to see an SESC badge summarizing the enterprise-level achievement of a core set of software engineering practices [Moore95].

Using This Book

Recognizing the unique nature of every enterprise, this text "slices and dices" the important collections of software engineering standards in a variety of ways. The chapters describing the collections usually organize their constituent standards in ways suggested by the creators of the collections. The chapters describing the context of software engineering allow one to select software engineering standards by building on recognized contextual strengths of an organization, or by remedying notable contextual weaknesses. The chapters describing the objects of software engineering present the opportunity to consider standards addressing a single object, such as process, of the discipline.

Throughout, emphasis is placed on the idea that two extreme policies should be avoided: (1) software engineering standards are not isolated islands of practice that should be individually adopted; and (2) software engineering standards (even a single SDO's collection) are not a monolithic whole whereby a commitment to one requires a commitment to all of them. Instead, the book provides a middle course, allowing the selection of coherent subsets of standards, suitable for the achievement of goals specific to an organization.