Chapter 3: An Overview of the IEEE Software Engineering Standards

The IEEE Software Engineering Standards collection is so substantial that it might appear impenetrable to even experienced software professionals. It is difficult to gain an overview, and the standards collection itself has not yet fully addressed this issue.

Older editions of the collection were published as a single volume, with the standards simply included in ascending numeric order, which unfortunately did not result in any particularly functional organization. The 1997 Edition had grown to the size of a metropolitan telephone book and with its heavier paper could certainly have inflicted bodily injury if inappropriately dropped! The Introduction to the 1997 Edition was so brief that it provided little more than a list of the standards; it really did not help the reader gain an overview.

The 1999 Edition was divided into four volumes organized according to a new framework description of the standards collection. Although well suited for systematic classification of the standards, the framework is complex and might be difficult for new readers to understand. The framework specifies the IEEE publication Software Engineering Standards, A User's Road Map [Moore97] as the "Overall Guide" for the standards collection. This excellent book further expounds on the framework model, and adds a comparative analysis of other standards outside of the IEEE collection. However, it would be more useful to an established expert on standards than to a neophyte user of the IEEE standards.

This chapter provides information complementary to the Introduction section of the standards collection, written for someone who is just becoming familiar with the IEEE standards. The SESC (the IEEE Software Engineering Standards Committee) framework model is described first, followed by a simpler organizational model of the standards. Next, interrelationships between key standards are discussed. The applicability of individual standards is then analyzed as a function of the criticality of the software application, the organization's existing process maturity, and other factors. The role of the top-level standard in the IEEE collection, IEEE/EIA 12207.0, is outlined. This standard is discussed in more detail in Chapter 4, "Software Life Cycle Processes." Finally, a brief discussion is provided on some topics not yet covered by the IEEE standards collection.

The SESC Framework

The Introduction section of the 1997 Edition consisted of only a brief Historical Perspective, followed by a Synopses of the Standards, with a single paragraph statement of scope and purpose for each standard. Other than mentioning IEEE Std. 730 as the historically first standard, the Introduction gave the reader no overview of how the different standards fit together, and no instructions on which standard a new reader should start with.

The Introduction to the 1999 Edition has been significantly expanded, relative to earlier versions. The Introduction succinctly states:

"The SESC collection has grown large enough that users can no longer be expected to intuitively perceive the relationships among the various component standards."

To address this problem, the SESC created a framework to describe the structure of the standards collection. This framework is described in the Introduction to the 1999 Edition, and detailed further in Software Engineering Standards, A User's Road Map [Moore97].

The framework specifies six layers and—for the middle three layers—four stacks, as pictured in Figure 3.1.

Figure 3.1
The SESC framework model.

The six layers are defined as follows:
1. Terminology: Documents prescribing terms and vocabulary

2. Overall Guide: One document providing overall guidance for the entire collection

3. Principles: Documents that describe principles or objectives for use of the standards in the collection

4. Element Standards: Standards that are the basis for conformity

5. Application Guides and Supplements: Guides and supplements that give advice for using the standards in various situations

6. Toolbox of Techniques: Descriptions of techniques that might be helpful in implementing the provisions of the higher-level documents

The four stacks correspond to the four objects in the object model of software engineering shown in Figure 3.2.

Figure 3.2
The SESC object model of software engineering.

1 Figure 5 – "Objects of software engineering" from the Introduction of the IEEE Standards, Software Engineering, 1999 Edition

The object model categorization of the standards might be elaborated as follows:

- Customer Standards: Standards that describe the interaction between the customer and supplier of a software engineering project

- Process Standards: Standards that describe processes spanning the life cycle of a software product or service, including acquisition, supply, development, maintenance, operations, and measurements

- Product Standards: Standards that explain the requirements for classes of software products—characteristics, measurements, evaluations, and specifications

- Resource Standards: Standards that recommend proper documentation, methods, models and tools for a well-managed software program and its related processes

Most of the standards are classified according to the framework as Element Standards in one of the four object categories. The four volumes are organized mostly according to the object model, with the layers not affected by this categorization distributed so as to even out the physical size of the volumes. Appendix A lists the standards contained in each of the volumes for convenience. Figure 3.3 summarizes the classification of individual standards in the SESC framework.

Figure 3.3
Classification of standards in the SESC framework.

SESC Alternative Organization of the Standards

A previous SESC classification of the standards is still listed as an alternative in the Introduction to the 1999 Edition. In some ways, this organization is easier to understand, which is why it is still included. The first four categories of the alternative model might be directly mapped to the framework model, as follows:

- Terminology: Maps to the Terminology layer of the framework model.
• Life Cycle Processes: Maps to the Process stack of the framework model. However, it only represents a subset of the Process stack.

• Tools: Maps to the Resource stack of the framework model. However, it only represents a subset of the Resource stack, corresponding to the Tools and Environments sub-stack.

• Reuse: Maps to the Resource stack of the framework model. However, it only represents a subset of the Resource stack, corresponding to the Reuse Libraries sub-stack.

• The other parts of the alternative model do not map directly onto the framework model. Each of the remaining sections contains standards from multiple parts of the framework model. The remaining parts of the alternative model are as follows:

  • Project Management: Includes IEEE Std. 1058 (Software Project Management Plans) and IEEE Std. 1490 (Guide to the Project Management Body of Knowledge), classified as Process standards by the framework model. However, it also includes IEEE Std. 1044 (Classification for Software Anomalies), which is classified in the generic Toolbox of Techniques layer by the framework model.

  • Plans: Includes IEEE Std. 730 (SQA Plans), IEEE Std. 828 (SCM Plans), and IEEE Std. 1012 (Software V&amp;V) from the framework model’s Process stack. Also includes IEEE Std. 1228 (Software Safety Plans) from the framework model’s Customer stack.

  • Documentation: Includes IEEE Std. 829 (Software Test Documentation), IEEE Std. 830 (Software Requirements Specifications), and IEEE Std. 1016 (Software Design Descriptions) from the framework model’s Resources stack. Also includes IEEE Std. 1063 (Software User Documentation) from the framework model’s Product stack. Also includes IEEE Std. 1233 (System Requirements Specifications) and IEEE Std. 1362 (Concept of Operations) from the framework model’s Customer stack.

  • Measurement: Includes IEEE Std. 982.1 (Measures for Reliable Software) and IEEE Std. 1061 (Software Quality Metrics Methodology) from the framework model’s Product stack. Also contains IEEE Std. 1045 (Software Productivity Metrics) from the framework model’s Process stack.

Problems with the SESC Models

The existing IEEE Software Engineering Standards were not developed with the SESC framework in mind. Instead, the SESC framework was developed later as an attempt to provide a systematic classification schema for a large, pre-existing set of standards. The standards don’t always neatly fit into the framework model. This is evidenced by numerous standards having secondary assignments. For example, IEEE Std. 829 (Software Test Documentation) has a primary classification as a Resource standard, and a secondary classification as a Process standard. Similarly, IEEE Std. 730 (Software Quality Assurance Plans) has a primary classification as a Process standard, and a secondary classification as a Product standard. In total, there are 15 such secondary classifications of standards and guides.

The SESC alternative model suffers the same classification ambiguity problem. For example, IEEE Std. 1058 (Software Project Management Plans), is assigned to the Project Management category, however, being a plan, can also be in the Plans category. Similarly, IEEE Std. 1012 (Software Verification and Validation) is assigned to the Plans category, but, implicitly defining a software life cycle model, and explicitly defining the supporting verification and validation tasks for the process, can also be included in the Life Cycle Processes category.

A more important short-coming in both of these models is that they don’t tell you how to get started, or how to apply the standards. The models don’t specify a phase-in strategy, and they don’t specify adapting the standards to your organization’s needs. The models don’t specify core standards for just getting started, and they don’t recommend standards based on your organization’s size, current process maturity, or application domain criticality.
For example, for your next project, do you need

- A separate *Software Project Management Plan*, according to IEEE Std. 1058?

- A separate *Software Configuration Management Plan*, according to IEEE Std. 828?

- A separate *Software Verification and Validation Plan*, according to IEEE Std. 1012?

- A separate *Software Quality Assurance Plan*, according to IEEE Std. 730?

After all, the template for a *Software Project Management Plan* includes sections for *Configuration Management, Verification and Validation*, and *Quality Assurance*. Under what conditions should you create separate plans, and when should you put all the information together into a single plan? Unfortunately, no guidelines are provided.

Admittedly, some of the individual standards provide recommendations for some questions of this type. For example, IEEE Std. 730 (SQA Plans) states that it "applies to the development and maintenance of critical software." It then goes on to specify the following minimum documentation:

- Software Requirements Specification

- Software Design Description

- Software Verification and Validation Plan

- Software Verification and Validation Report

- User Documentation

- Software configuration Management Plan

Thus, after you have read and understood a number of standards, you might conclude that, if you have a critical software application ("Software whose failure would impact safety or cause large financial or social losses"), you should follow, at a minimum, the following standards:

- IEEE Std. 730: Software Quality Assurance Plans

- IEEE Std. 828: Software Configuration Management Plans

- IEEE Std. 830: Software Requirements Specifications

- IEEE Std. 1012: Software Verification and Validation

- IEEE Std. 1016: Software Design Descriptions

- IEEE Std. 1063: Software User Documentation

Further investigation would, however, reveal that there are a number of other IEEE Software Engineering Standards, which should rightfully be considered as mandatory for critical software, including:

- IEEE Std. 829: Software Test Documentation
IEEE Std. 1008: Software Unit Testing

IEEE Std. 1028: Software Reviews

IEEE Std. 1058: Software Project Management Plans

IEEE Std. 1228: Software Safety Plans

The problem here is simply that the SESC models, and the Overall Guide layer\(^2\), fail to provide the reader with information about which standards can be considered mandatory or appropriate for their organization or project.

\(^2\)Software Engineering Standards, A User’s Road Map [Moore97]. It should be mentioned that IEEE/EIA 12207.0 (Software Life Cycle Processes) has some direct bearing on this issue. This standard specifies primary, supporting, and organizational process requirements, and guidelines for adapting these to your organization or project. This standard is discussed in detail in Chapter 4.

A Simplified Organizational Model

This section presents a simplified organizational model of the IEEE Software Engineering Standards. This model restricts itself to a subset of the standards collection, concentrating exclusively on what the author considers to be the core standards. With one exception (IEEE Std. 1042), only the Element Standards (those which “are the basis for conformity”) are considered. System engineering standards are excluded, because they pertain to a closely related but distinct field. Other standards are excluded because, although useful, they are not amongst the most essential.

We start with a simplified object model of software engineering, as shown in Figure 3.4.

Figure 3.4
A simplified object model of software engineering.

The objects in this model might be described as follows:

- **Process**: Standards, descriptions, and other means by which an organization describes the process to be used for multiple software projects. Activities for individual projects are excluded.

- **Project**: A software project is a complex entity, consisting of people working together for a prolonged period of time, using tools, facilities, and other resources, to create a software product. Plans and reports of project activities are meant to be included as an element of a project, but work products, such as the software product and specifications, are excluded.

- **Work Product**: The software product itself, user documentation for it, and intermediate documents produced during the project to specify, describe, or document the software are all meant to be included in this object category. Examples of intermediate documents include Software Requirements Specifications and Test Case Specifications, although these will not be delivered to the end user. Documents describing project activities are not meant to be included in this category. Work products might be conventional documents, or they might represent electronic records, or be sections of larger documents. The *Introduction to Volume Four: Resource and Technique Standards* refers to this type of object as a "Process Information Product" or "PIP" for short, but this term is not used here because it is too cryptic.

Next, we divide software engineering activities into the following two categories, which might correspond approximately to organizational units charged with performing these activities:

- **Development**: Activities “by which user needs are translated into a software product.” (IEEE Std. 610.12,
definition for Software Development Process). These activities are meant to include "translating user needs into software requirements, transforming the software requirements into design, implementing the design in code..." However, activities that are intended to evaluate the development activities or their associated work products are not meant to be included here.

- Verification and Validation: Software V&V is defined by IEEE Std. 610.12 as: "The process of determining whether the requirements for a system or component are complete and correct, the products of each development phase fulfill the requirements or conditions imposed by the previous phase, and the final system or component complies with specified requirements." Software Quality Assurance activities, a broader concept, are also meant to be included, in order to include IEEE Std. 730.

This categorization of activities reflects IEEE Std. 1012 (Software Verification and Validation), which also differentiates between Development versus V&V Activities and Outputs. This standard defines V&V Inputs as the outputs of either the Development Process or of V&V Activities from previous project phases. The V&V Activities executed in a given project phase produce additional V&V Outputs for that phase.

Combining these two classifications, we can organize the most important IEEE Software Engineering Standards as shown in Figure 3.5.

**Figure 3.5**
A simplified classification of the standards.

Note that this simplified model still requires that certain standards be assigned multiple slots. However, in each case, we can clearly specify which subsections of these standards belong exclusively to which category. For example, the Test Plan, Test Item Transmittal Report, Test Log, Test Incident Report, and Test Summary Report sections of IEEE Std. 829 (Software Test Documentation) represent Project Documentation standards, whereas the Test Case Specification and Test Procedure Specification sections represent Work Product standards. The first set of test documents (plans and reports) describes planned and actual activities for the particular project; the second set of test documents (specifications) describes expected behavior of the software product.

The next section will explore interrelationships between the key standards listed previously, based on project information flow. The following section provides guidelines on applicability of the standards using a maturity model.

**Information Flow Between Documents Specified by Core Standards**

The IEEE standards call for certain documents to be produced. These might represent plans, reports, specifications, or other types of documents. These might be conventional documents, electronic records, or sections of larger documents. The information flow between these documents defines an implicit relationship between the standards governing those document types. The specifics will depend on the individual project, but consider the information flow model shown in Figure 3.6.

**Figure 3.6**
Document information flow.

This diagram uses the following list of acronyms:

<table>
<thead>
<tr>
<th>PHAR</th>
<th>Preliminary Hazard Analysis Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDD</td>
<td>Software Design Description</td>
</tr>
<tr>
<td>SPMP</td>
<td>Software Project Management Plan</td>
</tr>
<tr>
<td>SRR</td>
<td>Software Review Report</td>
</tr>
</tbody>
</table>
SRR is not a standard acronym, but is meant to represent a report for any of the types of software reviews specified by IEEE Std. 1028, including Audit Reports, Inspection Reports, Management Review Reports, Technical Review Reports, and Walk-through Reports. This standard and the five types of software reviews are discussed in more detail in Chapter 6, "Process Standards."

This diagram might seem complex if your organization does not yet produce much software documentation, but at the same time it is greatly simplified relative to complex projects for critical applications. As a note, IEEE/EIA 12207.1 provides material useful in building an information model for complex projects.

It is worth mentioning that system-level documentation is not shown, but that a System Requirements Specification (IEEE Std. 1233) and other system level plans, specifications, and descriptions are logical inputs to the SSP, PHAR, SPMP, SRS, and other software documents. Software requirements are derived from system requirements, and, ideally, should be traced to system requirements.

User Documentation, governed by IEEE Std. 1063, is not shown on the diagram. The HAR and SRS are both logical inputs for the User Documentation, which in turn might be considered input for the TPS documents. This was omitted only to keep the diagram simple.

Software reviews can be effectively used for any of the software documents shown in the diagram, not only for the SRS, SDD, and code. The number and type of software reviews can be specified by the SVVP.

An SCMP (Software Configuration Management Plan, IEEE Std. 828) and an SQAP (Software Quality Assurance Plan, IEEE Std. 730) are not included in the diagram. This was done to simplify the diagram, not to imply that these are unimportant. These (and the SVVP, and even the SSP) can be written as sections of the SPMP. This would be expeditious if only one individual is responsible for maintaining the plans for all these activities.

Finally, the test documentation schema shown in Figure 3.6 follows the full set of types of test documents specified by IEEE Std. 829. The author has found this to be an excellent, modular style of documenting testing activities, but many organizations use a simplified test documentation schema. This is discussed further in a Chapter 5, "Document Standards."

Applicability of Standards
This section discusses which IEEE software engineering standards are applicable to your organization or project under which circumstances. Included in this concept is an incremental phase-in of the standards, because it is unrealistic to expect to use all of them at once. The guidelines provided here are not meant to be definitive. You must carefully analyze laws, regulations, guidelines, and other standards specific to your application domain to determine the minimum acceptable process model. Figure 3.7 provides guidelines for implementing specific standards based on the criticality of the application, the size of the project team, the existing process maturity of the organization, and any external requirements for the process.

Figure 3.7
Pyramid of applicability—a simple maturity model.

There is an obvious and direct relationship between the number of IEEE Software Engineering Standards successfully adopted by an organization and the software process maturity of that organization. By specifying a rational and orderly adoption sequence for the standards, the Pyramid of Applicability in Figure 3.7 provides a simple Maturity Model. This can be used to assess an organization's maturity, and plan software process improvements.

As the criticality of the application increases, the number of standards that can be followed increases also. The criticality is a function of both the severity and probability of hazards and errors. The Software Integrity Level concept defined in IEEE Std. 1012 is a synonym for criticality, and that standard specifies the minimum set of V&V tasks that can be performed as a function of the software integrity levels. The terms used in the diagram are borrowed from IEEE Std. 1012, with "None" referring to a software integrity level 0. Figure 3.6 is not intended to recommend foregoing activities unless the stated criticality is reached. The information flow diagram is presented only to recommend full conformance with the IEEE standard for an activity if that criticality is reached. For example, Management Reviews should be performed even for Low criticality software, even if these are not yet conducted in accordance with IEEE Std. 1228 for Software Reviews.

As the size of a team grows, the more need it has for standards to manage the project activities. Conversely, if more standards are needed for other reasons, more personnel might be required, for example, to support more independent review and testing activities. Figure 3.7 is not meant to imply that, if you can develop the software with just a few people, you can ignore most of the software engineering standards. Management must take responsibility for the appropriateness of the software process, and provide adequate personnel and other resources to support the process.

The existing process maturity of the organization will limit what set of standards can reasonably be expected to be implemented successfully. As described in Chapter 2, "Guidelines for Software Process Improvements," changes to the process should be approached incrementally. It is unrealistic to expect to introduce more than a few of the IEEE Software Engineering Standards at once.

However, external requirements might dictate attributes of the software process, and might force an organization to make a larger change to its process. For example, the Quality System Regulation affecting medical device manufacturers in the United States provides hard requirements for the process of developing and validating medical device software. The IEEE standards collection can help make compliance more achievable, both because the IEEE standards are recognized and respected by regulatory bodies such as the FDA, and because adoption of the IEEE standards means that the substantial effort to develop such standards does not have to be performed in-house.

The following subsections briefly describe the process model at each level in the maturity model. The levels of this maturity model are distinct from the levels of the SEI CMM model, and there is no intent to imply otherwise. The reader is also referred to the summary descriptions of the standards provided in the Introduction to the 1999 Edition.

Level 0

Level 0 in the maturity model corresponds to adoption of high-level processes, but the absence of any formalized element standards. This level is appropriate only if there is absolutely no criticality to the software, and there are no external requirements for additional standards.
IEEE/EIA 12207.0 provides appropriate guidance at maturity level 0. This standard exists at the "principles" layer according to the SESC framework model.

IEEE/EIA 12207.0 establishes a common framework for all software life cycle processes, as follows:

- Primary processes, including acquisition, supply, development, operation, and maintenance
- Supporting processes, including documentation, configuration management, quality assurance, verification, validation, joint review, audit, and problem resolution
- Organizational processes, including management, infrastructure, improvement, and training

Adoption of IEEE/EIA 12207.0 will help clarify the tasks that should be performed, for example, during the software development process. It does not provide much specific guidance on how to perform some of these tasks. For example, it specifies that "the developer shall establish and document software requirements," and that these should include "functional and capability specifications" and "interfaces external to the software item," but does not specify the format or detailed information content of a Software Requirements Specification. For this, the element standard IEEE Std. 830 (IEEE Recommended Practice for Software Requirements Specifications) is needed.

This level allows adopting simpler process models than lower levels. For example, the V&V processes of IEEE/EIA 12207.0 are far simpler than those of IEEE Std. 1012. One could use the V&V processes of IEEE Std. 12207.0 as a first step and later adopt the more detailed process provided by IEEE Std. 1012. As another example, even at a low level of maturity, organizations need some semblance of configuration management. The IEEE/EIA 12207.0 configuration management process is good enough until one gets to the higher level of maturity and applies IEEE Std. 828 (IEEE Standard for Software Configuration Management Plans).

IEEE/EIA 12207.0 is discussed in more detail in Chapter 4.

This level of process maturity should not be unduly ridiculed. The author has seen sophisticated, complex software applications developed with user documentation complemented only by non-standardized and minimal design notes and test reports. Most project activities were executed informally, and without being documented. The high quality of the final software bore tribute to the abilities, dedication, and survival of the project team. The problem with this approach is that individuals become indispensable, and project successes are not easily reproduced. For critical applications, an absence of a process model is simply unacceptable.

**Level 1**

Level 1 in the maturity model corresponds to the simplest process for a software project beyond the "black box" of Level 0. A *Software Requirements Specification* is written to document what the software should do, and *Test Documentation* is created to demonstrate that the software actually does this.

This level is only suitable for low criticality applications, and low external requirements for the process. This level is recommended for a first attempt to formalize the software process.

IEEE Std. 830 for Software Requirements Specifications defines a standard for developing complete and unambiguous specifications of the software’s functional characteristics. The SRS is probably the single most important software document. The SRS represents an agreement between the client and development groups on the software. The SRS instructs the software engineers as to what they shall design. The SRS is an essential input document for developing tests. The SRS helps determine acceptance of the software product at the end of the development process. The author has found IEEE Std. 830 to work extremely well for a wide variety of applications, and to help inexperienced personnel learn to document software requirements quickly and effectively.

Writing a good SRS is a difficult task, which requires skill and experience. Too often, the design is described, instead of the requirements being specified. The SRS should document what, not how. This is easier said than done.
for many software engineers. If informal design documentation has already been created, do not be tempted to start with this to create an SRS. IEEE Std. 830 provides a flexible set of templates for creating a truly useful SRS.

IEEE Std. 829 for Software Test Documentation specifies a variety of documents and an implicit associated process model for software testing. Software testing is the single most basic and indispensable software V&V activity. If software is not tested, it can almost be guaranteed to not function correctly.

A project might specify a subset of the work products from IEEE Std. 829 in the Test Plan. The author has found the modular style for test documentation of this standard to be an excellent alternative to the more monolithic approach typical of proprietary techniques. At first, the large number of different types of documents appears overly complex, but over years of use the author has learned to appreciate this standard more and more. It is important to remember that conformance with this standard does not require producing all the document types for each project, and that the various documents might actually be written as sections of a single larger document if this is preferred.

At this level of process maturity, formalized software testing is probably restricted to system-level testing, because the lower-level software design units are not systematically specified.

The adoption of IEEE Std. 829 and IEEE Std. 830 represent a major process improvement over Level 0, and will require substantial effort to implement correctly.

Level 2

Level 2 in the maturity model formalizes several additional important parts of the software process beyond those of Level 1. Level 2 formalizes creation of a project plan, the documentation of the software design, and a systematic approach to reviewing work products.

This level of process maturity is suited to projects of moderate criticality, or moderate external requirements for the process model. It is recommended for a second phase of formalization of the software process.

The most important type of plan, the Software Project Management Plan, is formalized in IEEE Std. 1058. An SPMP helps ensure that a systematic process is defined for its project. In the absence of other plans, it should describe Verification and Validation, Software Quality Assurance, and Software Configuration Management activities, in Section 7 of the template defined in the standard. The author has found the SPMP template in IEEE Std. 1058 to be excellent, providing a comprehensive list of issues that, if worked out in detail and updated as the project progresses, provides all the information needed to effectively plan and manage a project.

The Software Design Description (IEEE Std. 1016) documents the software design, both high-level and low-level. It describes how the software will address its requirements, in sufficient detail to enable the implementation of the software to proceed, and to enable testing of individual modules. Availability of a SDD thus allows for software testing to be performed more effectively at the integration and unit levels.

Software Reviews represent the second most important form of V&V activity besides testing. IEEE Std. 1028 provides a concise, yet complete description of five major types of software reviews. Technical Reviews, Walk-throughs, or particularly Inspections are invaluable design controls during early project phases. Management Reviews are essential for management to assume responsibility for a project in a meaningful way, and should be part of any project. Audits are important as a quality assurance technique, to help enforce process compliance.

Level 2 represents a major software process improvement relative to Level 1. It is not recommended to attempt more than this combination at once.

Level 3

Level 3 in the maturity model extends Level 2 with standards for software quality assurance, verification and validation, configuration management, and software safety. Each of these standards provides templates for writing plans. These standards imply additional project activities, beyond simply creating the plans. For example, the configuration management standard requires that a systematic process be defined for requesting, evaluating,
approving or disapproving, and implementing changes to a controlled work products (Configuration Items). Similarly, the V&V standard specifies in detail a wide variety of V&V activities to be performed in different project phases, as a function of the Software Integrity Level determined near the beginning of the development process.

The level of process maturity is suited to major criticality applications, or for when major external requirements exist for the process. This level is recommended for a third phase of process formalization.

IEEE Std. 730, the first of the IEEE Software Engineering Standards, provides a template for creating Software Quality Assurance Plans. This standard applies to critical software, and states that a subset of the requirements might be applied to non-critical software. For critical software, it specifies minimum documentation requirements for the project, including:

- Software Requirements Specification (SRS)
- Software Design Description (SDD)
- Software Verification and Validation Plan (SVVP)
- Software Verification and Validation Report (SVVR)
- User Documentation
- Software Configuration Management Plan (SCMP)
- Other documentation, including a Software Project Management Plan (SPMP)

It is reassuring to note that, by Level 3, we have included standards for each of these types of required documents.

IEEE Std. 1012 describes Verification and Validation activities to be performed in a project as a function of the Software Integrity Level. We recall that V&V is defined by IEEE Std. 610.12 as:

> The process of determining whether the requirements for a system or component are complete and correct, the products of each development phase fulfill the requirements or conditions imposed by the previous phase, and the final system or component complies with specified requirements.

The software integrity level represents the criticality of the software, and a scale of 0 to 4 is defined in the standard. Minimum V&V tasks are specified for the different integrity levels, and are organized into activity groups. V&V activities are shown in a waterfall Software Life Cycle Model, but may be mapped to other models. The standard also provides templates for writing a Software Verification and Validation Plan, and a Software Verification and Validation Report. Software testing and review activities form the core of the V&V activities.

Std. 1012 is a large and complex standard, and will appear daunting at first. It lists a large number of project activities and work products, which might seem to be too much relative to your previous projects. The 1998 revision of this standard has helped this matter significantly, because it provides guidelines for tailoring the V&V activities according to the integrity level. This mechanism enables focusing limited project resources on the V&V activities that can provide the most benefit, while still maintaining conformance with the standard. V&V activities can benefit the project, in terms of quality gains, and reduced overall cost and schedule. V&V should not result in unproductive busywork!

IEEE Std. 1012 draws on IEEE Std. 829 for Software Test Documentation, and IEEE Std. 1028 for Software Reviews, because they provide more detailed instructions for performing these two fundamental V&V techniques. IEEE Std. 1012 is a higher-level standard, which allows you to plan and execute your project V&V activities in a more systematic manner. For example, even if you were performing software reviews at Level 2, at Level 3 you now have specific requirements for what reviews must be performed as a function of the software criticality.
IEEE Std. 828 for Software Configuration Management Plans specifies a formal approach to identifying and controlling changes to project work products, including the software itself. The controlled work products are referred to as **Configuration Items**. The configuration items must be identified, and changes to the items, controlled. A **Configuration Control Board** (CCB, also sometimes called a "Change Control Board") is required for the critical task of approving or disapproving changes to configuration items. This very flexible standard enables a CCB to be an individual or a group, and allows for multiple CCBs.

For many organizations, the change from an informal approach to decision making (approving or disapproving changes) to an orderly process can be profound. With an informal process, in some organizations, an autocratic manager might make all decisions, in other organizations a democratic consensus building approach might be required. The most ineffective technique, found all too often, consists of opposing factions struggling against each other in a harsh Darwinian contest to enforce their point of view. A formalized change control process helps transform critical project decision making from a disruptive, unpredictable headache to a civilized, manageable task.

IEEE Std. 1228 for Software Safety Plans is intended to improve the safety of critical software. It calls for identification and analysis of hazards, and the analysis of the software requirements, design, code, testing, and change activities relative to these hazards. The **Risk** assigned to hazards in this standard corresponds to the **Software Integrity Level** from Std. 1012. The **Criticality Analysis** specified by Std. 1228 is an important task of correlating the system risks to the software, and thus driving the scope of the V&V activities. If you have a safety critical software application, you should follow this standard.

**Level 4**

Level 4 in the maturity model represents conformance with all key IEEE Software Engineering Standards listed in Figure 3.5. It corresponds to a mature process, and is suited to high criticality software applications.

IEEE Std. 1074 is a standard for developing a **Software Life Cycle Process** (SCLP). The SLCP is project-specific, and must be derived from a **Software Life Cycle Model** (SLCM) to be selected by the organization. The standard does not specify any standard SLCMs, but defines a set of Activities organized into different **Activity Groups** that must be mapped to the SLCM. This standard is useful not only for generating the SLCP, but also for independent assessment of the adequacy of the SLCP. Use of this standard will help ensure that the project plan and other related plans are sufficiently comprehensive for their intended purpose.

IEEE Std. 1042 (**IEEE Guide to Software Configuration Management**) is classified by the **Introduction to the 1999 Edition** as an Application Guide rather than an Element Standard. It might thus be viewed as an elaboration of IEEE Std. 828 for Software Configuration Management Plans. It is the only guide to be included in the core standards group, because it provides substantial additional information beyond IEEE Std. 828. Application of IEEE Std. 828 in conjunction with the guidelines provided by IEEE Std. 1042 can reasonably be expected to yield a more substantial and mature software configuration management process than use of IEEE Std. 828 alone.

IEEE Std. 1008 provides a standard for performing **Software Unit Testing**. Unit testing is defined by IEEE Std. 610.12 as "**Testing of individual hardware or software units or groups of related units.**" The term "module" is often used interchangeably with the word "unit" and might be more familiar to the reader. Unit-level testing is performed on individual software modules or groups of software modules, and is an important technique for validating critical software applications. Critical software requirements should be traced through the design to individual modules in the implementation, and unit testing should be focused on these critical software modules. IEEE Std. 1008 in no way supercedes IEEE Std. 829 for software test documentation.

IEEE Std. 1061 for a **Software Quality Metrics Methodology** describes how to institute quantifiable quality measures for software. It does not define specific metrics as IEEE Std. 982.1 does. Instead, it describes a process for establishing software quality requirements, and identifying, implementing, analyzing, and validating software quality metrics for those requirements. IEEE Std. 982.1 and IEEE Std. 1061 can be used in combination.

IEEE Std. 982.1 (**IEEE Standard Dictionary of Measures to Produce Reliable Software**) provides a catalog of both process and product measures used as indicators of reliability. **Software Reliability** as defined in this standard ("**The probability that software not cause the failure of a system for a specified time under specified conditions.**")
represents an effective measure of the more general concept of Software Quality. This standard might be used effectively in combination with IEEE Std. 1061. Some of the measures might be applied during software reviews prior to testing, for example, for Code Inspections.

Level 5

Level 5 in the maturity model represents conformance with the entire set of IEEE Software Engineering Standards. This corresponds to a high level of process maturity, and is suited to high criticality software applications, even for very large organizations. Because the IEEE standards are continually revised, and new standards are added, an ongoing program is required to maintain conformance with all the standards.

Level 5 is outside the scope of this book. If you are ready for this level, you don’t need to be reading this book, except perhaps for your personal amusement.

Missing Standards

The IEEE Software Engineering Standards collection is a dynamic body of knowledge, which is growing at a fast pace. One of its goals is to remain sufficiently generic so that each organization can use new techniques and tools, while still maintaining conformance. Stylistic preferences are excluded from the standards collection, but, at the organizational level, must finally be decided. Several important types of standards must be provided by the organization to supplement the IEEE standards, including:

- **Programming Standards.** Organizations are expected to maintain standards for the development of the software source code. Their purpose is to increase software quality, by proper commenting, limiting module complexity, systematic naming conventions, and other techniques. Such standards are often dependent on the choice of programming language, and might represent stylistic preferences of the team members. It is important to an organization to standardize these conventions, to allow for more effective use of multiple software engineers.

- **Design Standards.** Organizations may also benefit from design standards. These can help ensure that consistent techniques are used, for example, in conjunction with object-oriented design methods. Guiding principles, such as encapsulation and information hiding, may be defined, and checklists may be developed to help with design reviews.

- **Criticality Rating Standards.** The risk ratings of IEEE Std. 1228 and the software integrity levels of IEEE Std. 1012 can be specified in more concrete terms for your application domain. The mapping can reflect any regulations or guidelines for your industry. For example, in the medical device industry, FDA guidelines for software development and validation are written in terms of the Level of Concern, which might be Major, Moderate, or Minor. To demonstrate compliance with the FDA guidelines using the IEEE standards, an appropriate mapping must be established between the FDA’s level of concern and the software integrity levels of IEEE Std. 1012.

- **Software Life Cycle Models.** The organization might want to define a standard Software Life Cycle Model, to provide standard consistent terminology for its software development process, and to serve as a template for the Software Life Cycle for each project. The IEEE standards do not specify a SLCM, even though IEEE Std. 1074 provides guidelines for how to select and adapt one. A number of competing models have been developed, and each organization should have the flexibility of selecting or developing its own model.

Summary

The most effective way to gain a working knowledge of the IEEE Software Engineering Standards is to restrict the focus to a few standards at a time. The emphasis should be to attain a practical, usable understanding of several standards, without getting distracted by the collection as a whole. The aim of this book is to help the reader get started with the core standards in a pragmatic manner.

Several key concepts have been presented in this chapter:
The SESC framework and alternative models

A simplified organization model for the standards collection

A list of the most important standards to get started with

An information flow model showing the relationship between the core standards in terms of project activities

A simple maturity model providing guidelines for the applicability of specific standards based on the criticality of your application domain, existing process maturity of your organization, and other factors.

Chapter 4, "Software Life Cycle Processes," will provide the context for further understanding of the individual standards.

© Copyright Macmillan USA. All rights reserved.