Course: Software Engineering
Multimedia University Cyberjaya,

ANSWERS TO ASSIGNMENT NO. 8

1. In Reading-61 on this course website, we provided Java software code that actually was derived from a UML2 class diagram. Working in reverse, you are to produce that UML2 diagram.
   To assist you in your job, we provided below the URS (User Requirement Specifications) that produced the UML2 diagram. These URS statements are actually, the “twice-reverse” or “two-step-backwards” of the Java software code.

   (a) The software application is about a Marina Club keeping track of boats moored at its berthing facility in its database register. The facility handles two types of boats: Power Boat and Sail Boat.
   (b) Each boat has information covering a state registration number, boat length, boat manufacturer and the year of manufacture.
   (c) Each power boat has information regarding the number of engines and the fuel type for the engines.
   (d) Each sail boat has information regarding the number of sails, the motor type that powered the sails (raise up and down) and depth of keel (structure of the boat bottom for stability in water).
   (e) The information for each boat must be fully encapsulated.
   (f) There must be a functionality to display the full information about each boat by just calling its state registration number.
   (g) The application also requires the functionality for new boats arriving to berth at the club to be added into the database register.

   PLEASE SEE LAST PAGE FOR ANSWER TO QUESTION 1 (PAGE 16 OF 16)

2. Capability Maturity Model® Integration (CMMI) is a process improvement approach that provides organizations with the essential elements of effective processes. It can be used to guide process improvement across a project, a division, or an entire organization. CMMI helps integrate traditionally separate organizational functions, set process improvement goals and priorities, provide guidance for quality processes, and provide a point of reference for appraising current processes.

   The Good Book addressed this subject very early (Chapter 2 on page 59). There was also big news on CMMI in Malaysia (Readings 47, 48, 49 and 50). During a job interview you were asked to explain in layman terms regarding what it means when one company in Malaysia is rated CMMI Level 5

   (In your explanations, do not exceed two paragraphs. The interviewer wants to know your basic understanding, not in very technical terms. If required, you may need to explain one or two. Hints: matured, improvement and processes.)

   ANSWER

   When a company is rated CMMI Level 5, it means that the company has instituted a process improvement activity that has reached Level 5 within its organization. The CMMI process being cumulative requires that the organization implements process
improvements in a step-by-step manner, successively achieving the rating of the lower levels up to the highest levels. For this company, it means the company has passed Levels 1, 2, 3, 4 and now reached Level 5. The CMMI is actually a measure of the performance matrix for ‘capability of the company’, its ‘maturity or gained experienced’ and its ‘integration to business operations’ in evolving products or services the organization provides to the market.

The implementation of CMMI can viewed from either the staged or the continuous approach (Reading-59, page 38 of 58). The contents of both approaches are the same (page 43 of 58), only the content organizations are different. The 5 different levels of maturity for the CMMI (Reading-49.pdf, page 41 of 58) are defined as follows:

1. **Initial** – process unpredictable, poorly controlled and reactive
2. **Managed** – process characterized for projects, is often reactive
3. **Defined** – process characterized for organization, is proactive
4. **Quantitatively Managed** – process measured and controlled
5. **Optimizing** – focused on continuous process improvement

3. The term Software Configuration Management Plan (SCMP) is the technical jargon for simply, the planning, monitoring and control of changes of software configuration items (CI) throughout the software life cycle (cradle to grave). Accordingly, the software product must change or evolve. Once in operation, defects are uncovered, operating environments change, and new user requirements surface.

In the chapter on Software Configuration Management Plan (SWEBOK Reading-62, page 105 of 202), it was stated:

“A discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements.”

(a) What is a configuration item (CI – very important) in this context?

**ANSWER**

A configuration item is “information” for an identified element that must be tracked, controlled and recorded to facilitate change and maintenance management. For example, a Software Configuration Item (SCI) is the “information” for a software element that must meet the above rule. Similarly, a hardware configuration item is the “information” for the particular hardware and its settings that must be tracked, controlled and recorded.

(b) The Good Book (Chapter 27, Change Management, page 776) looks at SCIs (software configuration item) and configuration objects differently. What is the difference? (Hint: Quote “If a change were made to the SourceCode object, the inter-relationships enable a software engineer to determine what other objects (and SCIs) might be affected.”)
From Figure 27.2 in the Good Book (page 777), we can see that a software configuration item (SCI) is a member of (contained inside) a configuration object. This is the difference. For example, a configuration object like “DesignSpecification” contains many software configuration items (SCI) namely: data design, architectural design, module design and interface design. These SCIs must be tracked, controlled and properly recorded to facilitate change and maintenance management of the software. The purpose of grouping software configuration items into appropriate configuration objects is to reduce the "overcrowding of information", and make it easier to pinpoint inter-relationships and connections. It is easier to see the inter-relationships made between configuration objects rather than software configuration items. (This is the point referred to by the quotation in this question).

(c) In one paragraph, describe what you understand by the Figure 27.4 in the Good Book (page 781) regarding the “layers of the SCM process”.

Since there are many complex activities to fully implement software configuration management covering all the items that collectively define the software, a layered process SCM description as shown in Figure 27.4 was introduced for the purpose. These layered processes ensure that any change occurring in a software configuration item will be caught in one or many of the processes and will "not escape unrecorded" or unmanaged. For example, a change in "name" of a piece of software code (e.g. Java class file) will be caught by the “Identification Layer” process for re-identification, will be recorded in the “Change Control Layer” process for date of change, who made the change, size of new source code file etc, will be recorded in the "Version Control Layer” process as to which software version the change was made (e.g. when many software versions have been released), will propagate to the "Auditing Layer” process to check the effect of this change to other related software configuration items and change them if necessary, and will activate the "Reporting Layer” process to update the appropriate software report logs to record that specific name change.

(d) In one paragraph, describe what you understand by the Figure 27.5 in the Good Book (page 785) regarding the “change control process”.

Figure 27.5 shows the series of steps, tasks or activities that a typical change to a software configuration item will have to go through. These steps depict the processes of controlling software changes beginning from the initial point of recognizing the need to conduct some changes to the software to the final point of either: "incorporating the change and distributing the new version” or “rejection, disapproval and denial of the change request.” The "change control process“ in this question essentially means the process for controlling any change in the software. You just do not go about making changes in software as you like as that would cause havoc, especially if the software is large and complex.
(e) What are the five categories (classes) of information that you need to document to build up the Software Configuration Management Plan, SCMP? What are the uses of each of the information? (Hint: the W5HH principle and IEEE Std 828 – 1998)

**ANSWER**

The 5 classes of information that we need to document to build up the Software Configuration Management Plan, SCMP (Reading-51C, page 8 of 22) are as follows:

1. **SCM Management** - *(Who?)* Information used to identify the responsibilities and authorities for accomplishing the planned activities
2. **SCM Activities** - *(What?)* Information used to identify all activities to be performed in applying to the project
3. **SCM Schedules** - *(When?)* Information used to identify the required coordination of SCM activities with the other activities in the project
4. **SCM Resources** - *(How?)* Information used to identify tools and physical and human resources required for execution of the Plan
5. **SCM Plan Maintenance** - *(How?)* Information used to identify how the Plan will be kept current while in effect.

(f) Putting the technical jargon aside, you were being asked, say during an interview, a simple question regarding the SCMP (Software Configuration Management Plan) you are to execute, to describe five of the many configuration items (CI) /information that you need to identify in a software project. Describe them.

(Hint: Configuration identification activities shall identify, name, and describe the documented physical and functional characteristics of the code, specifications, design, and data elements to be controlled for the project, page 10 of 22 IEEE Std 828 – 1998. The SCM Plan shall specify an identification system for assigning unique identifiers to each configuration item to be controlled. It shall also specify how different versions of each are to be uniquely identified. Identification methods could include naming conventions and version numbers and letters.)

**ANSWER**

Actually, there are so many configuration items (CI) that must be identified, documented, recorded and controlled. Normally, for large and complex software projects a specific department (Software Documentation / Software Configuration Control) is usually assigned just to perform this role.

For the purpose of answering this question, the 5 examples of configuration items are listed below:

1. **Data Design** – the blueprints of the database model, the database SQL scripts, the database structures and the relationships between the various information in the database tables.
2. **Module Design** – the source codes, blueprints (UML diagrams) for the Use-Cases, Class diagrams, Object diagrams, Sequence diagrams, Statechart diagrams, Deployment diagrams etc, including the software codes, authors etc.
3 Architecture Design – software codes installation directories, the application engines, the application layers, the version numbers, the application parameter settings, etc.

4 Third Party Software – the installation directories and setting parameters for the third party software if used in the project.

5 Hardware Configuration – the names, models, components of your hardware and the setting parameters of the hardware used in the project.

4. One of the main objectives of the Software Engineering Body of Knowledge (SWEBOK Reading-62, page 24 of 202) is to set a boundary regarding subject material that is recognized as being within the Software engineering discipline. It is organized into the first ten Knowledge Areas (KAs) listed in Table 1 below.

Table 1 The SWEBOK Knowledge Areas (KAs)

<table>
<thead>
<tr>
<th>Knowledge Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software requirements</td>
</tr>
<tr>
<td>Software design</td>
</tr>
<tr>
<td>Software construction</td>
</tr>
<tr>
<td>Software testing</td>
</tr>
<tr>
<td>Software maintenance</td>
</tr>
<tr>
<td>Software configuration management</td>
</tr>
<tr>
<td>Software engineering management</td>
</tr>
<tr>
<td>Software engineering process</td>
</tr>
<tr>
<td>Software engineering tools and methods</td>
</tr>
<tr>
<td>Software quality</td>
</tr>
</tbody>
</table>

Essentially, the 10 subject topics above define the coverage and subject matter of "Software Engineering". For this question, you are to provide a one paragraph description of “what you understand (try to express in your own words from reading your reference material – later may be in examinations)” by each of the subject topic above. An example is provided below:

(Hint: Read SWEBOK, summarize in your own words – e.g. I did that for Item 1 below using Page 26 and 30 of 202 in SWEBOK)

<table>
<thead>
<tr>
<th>NO</th>
<th>KNOWLEDGE AREAS IN SOFTWARE ENGINEERING</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Software Requirements</td>
</tr>
<tr>
<td></td>
<td>This knowledge area deals with information regarding the requirements of the software to be built following established software engineering principles. It covers the purpose of the software (e.g. to solve what real-world problems), the description of the product, the functionality of the product, the system requirements (environment in which the product will be deployed), the processes of gathering requirements information (user interviews, existing information, literature research), the analyses of the gathered information producing some conceptual models, subsequently producing documentation and specifications based on gathered information, validating or checking back the software requirements specifications with the client, users etc for their agreement, discuss and negotiate with the client where necessary on the practical considerations of implementation (e.g. phased approach), schedule (dates, times, resources and duration), costs etc.</td>
</tr>
</tbody>
</table>
2 **Software Design**  
This knowledge area deals with the translation of the software requirements into suitable design models using textual and graphical descriptions (ERD, DTD, UML2 diagrams, etc), software requirements analyzed to produce both the structural and behavioral descriptions of the software internal structure, the architecture, the components and the interfaces, the strategies of design and techniques (problem solving, abstraction, coupling, encapsulation, reusable design patterns, etc), data-oriented design, function-oriented design, object-oriented design, component-based design, software application layers, software application architectures, handling events, handling and error trapping, application concurrency, data persistence and storage, and deployment distribution.

3 **Software Construction**  
This knowledge area deals with the translation of software design into the production of software codes, by minimizing construction complexity for ease of understanding using software constructs like small routines, subroutines, functions, procedures, reusable components, etc, enforcing uniform coding and commenting styles to ensure ease of change, ease of testing and modifications, applying software coding standards and recommended practices, e.g. using specialized tools (Integrated Development Environments – IDE, syntax checker, code generators, unit testing tools, etc), using the right and flexible language for construction, using established algorithms and patterns, using ready made and available objects and software libraries, developing standard user interfaces, etc.

4 **Software Testing**  
This knowledge area deals with the testing of the software program for defects (bugs, errors, etc) and non-conformities. Technically testing of software is termed validation and verification of software: validation means testing to ensure that the user stated functionalities have been achieved (doing the right things) and verification for achieving absolutely correct results, implementing the correct algorithms, calculations or program flows (doing the things right). Software testing consists of the static and dynamic tests: static test for verification of the software code and dynamic test for the verification of the behavior of a program while it is running. This area also deals with testing tools (generation of test cases), testing techniques and testing objectives. Tests should be run on both the “black-box” and “white-box” methodologies for a finite set of test cases, suitably selected from the many and in fact infinite possibilities. Testing also covers unit tests, integration tests, system tests, performance tests, stress tests, acceptance tests, installation tests, etc.

5 **Software Maintenance**  
This knowledge area deals with the totality of activities to support the software in a cost-effective manner through the entire software life cycle. Pre-delivery activities include planning for post-delivery operations, for maintainability, and for logistics determination for transition activities. Post-delivery activities include software modification, training, and operating or interfacing to a help desk. Software maintenance is also about the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment. It deals with the need for maintenance, the management of maintenance (tracking and
control), cost of maintenance, type of software maintenance (preventive, corrective, proactive, reactive), the processes, categories of maintenance, contractual maintenance issues etc. For maintainability, software must be designed with ease of maintenance (easy to understand design and source codes), can be reversed engineered and forward engineered.

| 6 | **Software Configuration Management**  
This knowledge area deals with the management and planning of the software configuration management process, software configuration item identification, software configuration and change control, software configuration status accounting (reporting), software configuration auditing, and software release management (version control) and delivery. The configuration management process controls the evolution and integrity of the software product by identifying its elements (configuration items and objects), managing and controlling change, and verifying, recording, and reporting on configuration information. There are specialized tools to handle and capture these configuration information like the CVS (Concurrent Versioning System) and repository databases like the “Registry” for Microsoft Operating Systems software. This knowledge area also deals with the listing or library of all software implemented for the project. |

| 7 | **Software Engineering Management**  
Software Engineering Management can be defined as the application of management activities—planning, coordinating, measuring, monitoring, controlling, and reporting—to ensure that the development and maintenance of software is systematic, disciplined, and quantified. With respect to software engineering, management activities occur at three levels: organizational and infrastructure management, project management, and measurement program planning and control. The sub-areas for software engineering management deal with initiation and scope definition, software project planning, software project enactment, review and evaluation, closure or post-completion activities and software engineering measurement. |

| 8 | **Software Engineering Process**  
This knowledge area deals with processes at two levels: The first level encompasses the technical and managerial activities within the software life cycle processes that are performed during software acquisition, development, maintenance, and retirement. The second level is the meta-level, which is concerned with the definition, implementation, assessment, measurement, management, change, and improvement of the software life cycle processes themselves. The technical and managerial activities are covered in the first 7 topics in this table (Software requirements, design, construction, testing, maintenance and software engineering management). |

| 9 | **Software Engineering Tools and Methods**  
Software development tools are the computer-based or computer-aided (CASE) tools that are intended to assist the software life cycle processes. Tools allow repetitive, well-defined actions to be automated, reducing the cognitive load on the software engineer who is then free to concentrate on the creative aspects of the process. Tools are often designed to support particular software engineering methods, reducing any administrative load associated with applying the method manually. Like software engineering methods, they are intended to make software engineering more systematic, and they vary in
scope from supporting individual tasks to encompassing the complete life cycle. There are many CASE tools: Requirements Modeling tools (UML), Design and Construction tools (code generators), Testing tools, Maintenance tools, Configuration Management tools, Engineering Management tools (projects), Process Management tools and Quality Management tools. The software engineering methods cover Heuristic Methods, Formal Methods and Prototyping Methods.

10 **Software Quality**  
This knowledge area deals with software engineering culture and ethics, value and costs of quality, models and quality characteristics, quality improvement, quality assurance, software validation and verification, reviews and audits, quality requirements, software defects, quality management techniques and quality measurement techniques.
5. There are many IEEE Standards that cover various important aspects of Software Engineering. We need to know at least some basic information about their existence and the purposes of their existence (i.e. brief, purpose, coverage).

The following table lists the relevant IEEE Standards for Software Engineering (may not be exhaustive) and the first row information for IEEE Std 830 had been filled as an example. In this question, you are to fill in the remaining rows following the example in the first row. (Use Reading-51 as your reference)

<table>
<thead>
<tr>
<th>No</th>
<th>STANDARD DOCUMENTS</th>
<th>BRIEF DESCRIPTION (OWN WORDS)</th>
<th>COVERAGE/KEY SECTIONS</th>
</tr>
</thead>
</table>
For the software we want to build in a project, we use this standard to prepare the contents of our software quality assurance plans (SQAP). This standard provides the minimum requirements for the contents of the software quality assurance plans.

**PURPOSE OF SQAP:**
This section shall delineate the specific purpose and scope of the particular SQAP. It shall list the name(s) of the software items covered by the SQAP and the intended use of the software. It shall state the portion of the software life cycle covered by the SQAP for each software item specified.

The software configuration management (SCM) plans provide a structure for identifying and controlling documentation, code, interfaces, and databases to support all life cycle phases. The SCM also provides and product information concerning the status of baselines, change control, tests, releases, audits, etc.

The SCMP sections include the following:
1) Overview
2) References
3) Definitions and Acronyms
4) The Software Configuration Management Plan
5) Tailoring the Plan
6) Conformance to the Standard |
<table>
<thead>
<tr>
<th>PURPOSE</th>
<th>This standard establishes the minimum required contents of a Software Configuration Management (SCM) Plan which provides approaches to good software configuration management planning. This standard applies to the entire life cycle of critical software; e.g., where failure would impact safety or cause large financial or social losses. It also applies to non-critical software and to software already developed. The application of this standard is not restricted to any form, class, or type of software.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE Std 829 – 1998 IEEE Standard for Software Test Documentation</td>
<td>BRIEF</td>
</tr>
<tr>
<td></td>
<td>PURPOSE</td>
</tr>
<tr>
<td></td>
<td>The Software Test Documentation sections include:</td>
</tr>
</tbody>
</table>
| 4 | IEEE Std 830 – 1998 IEEE Recommended Practice for Software Requirements Specifications | **BRIEF**
The content and qualities of a good software requirements specification (SRS) are described and several sample SRS outlines are presented. This recommended practice is aimed at specifying requirements of software to be developed but also can be applied to assist in the selection of in-house and commercial software products.

**PURPOSE**
This recommended practice describes recommended approaches for the specification of software requirements. It is based on a model in which the result of the software requirements specification process is an unambiguous and complete specification document. It should help software customers to accurately describe what they wish to obtain; software suppliers to understand exactly what the customer wants; individuals to accomplish the following goals: develop a standard software requirements specification (SRS) outline for their own organizations; define the format and content of their specific software requirements specifications; develop additional local supporting items such as an SRS quality checklist.

| 5 | IEEE Std 1012–2004 IEEE Standard for Software Verification and Validation (V & V) | **BRIEF**
Software V&V processes consist of the verification process and validation process. The verification process provides objective evidence whether the software and its associated products and processes 1 Conform to requirements (e.g., for correctness, completeness, consistency, accuracy) for all life cycle activities during

|  | The Recommended Practice for SRS sections include: | 1 Overview
2 References
3 Definitions
4 Considerations for producing a good SRS
5 The Parts for an SRS
6 The Standard for Software V & V sections include:
1 Overview
2 References
3 Definitions
4 Software Integrity Levels
5 Software V&V Processes
6 Software V&V Reporting, Administrative and Documentation requirements
7 Software V&V Plan Outline

Page 11 of 16
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>each life cycle process (acquisition, supply, development, operation, and maintenance)</td>
</tr>
<tr>
<td>2</td>
<td>Satisfy standards, practices, and conventions during life cycle processes</td>
</tr>
<tr>
<td>3</td>
<td>Successfully complete each life cycle activity and satisfy all the criteria for initiating succeeding life cycle activities (e.g., building the software correctly)</td>
</tr>
</tbody>
</table>

**PURPOSE**

The purpose of this standard is to establish a common framework for V&V processes, activities, and tasks in support of all software life cycle processes, including acquisition, supply, development, operation, and maintenance processes, define the V&V tasks, required inputs, and required outputs, identify the minimum V&V tasks corresponding to a four-level software integrity scheme, define the content of a software V&V plan (SVVP).

---

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>

**BRIEF**

An SDD (Software Design Document) is a representation of a software system that is used as a medium for communicating software design information. This recommended practice is applicable to paper documents, automated databases, design description languages, or other means of description.

**PURPOSE**

This recommended practice specifies the necessary information content and recommends an organization for Software Design Descriptions (SDDs). This document does not explicitly support, nor is it limited to, any particular software design methodology or descriptive technology. It

---

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The Recommended Practice for SDD sections include:</td>
</tr>
<tr>
<td>1</td>
<td>Overview</td>
</tr>
<tr>
<td>2</td>
<td>References</td>
</tr>
<tr>
<td>3</td>
<td>Definitions</td>
</tr>
<tr>
<td>4</td>
<td>Considerations for producing an SDD</td>
</tr>
<tr>
<td>5</td>
<td>Design description information content</td>
</tr>
<tr>
<td>6</td>
<td>Design description organization</td>
</tr>
</tbody>
</table>
will guide the production of anything from paper design documents to an automated database of design information. For an organization in the process of developing a design description standard, use of this document will help the new standard meet the needs of all of its users. For an organization with a mature design description standard, it should prove useful in evaluating and modifying that standard in light of the informational and organizational needs of the design description user community.

This standard defines five types of software reviews, together with procedures required for the execution of each review type. This standard is concerned only with the reviews; it does not define: procedures for determining the necessity of a review, nor does it specify the disposition of the results of the review. Review types include management reviews, technical reviews, inspections, walk-throughs, and audits.**

**PURPOSE**
This standard is meant to be used either in conjunction with other IEEE software engineering standards or as a stand-alone definition of software review procedures. In the latter case, local management must determine the events that precede and follow the actual software reviews. | The Standard for Software Reviews sections include:
1. Overview
2. References
3. Definitions
4. Management Reviews
5. Technical Reviews
6. Inspections
7. Walkthroughs
8. Audits |

A consistent way to measure the elements that go into computing software productivity is defined. Software productivity metrics terminology are given to ensure an understanding of measurement data for both source code and document | The Standard for Software Productivity Metrics sections include:
1. Overview
2. References
3. Definitions
4. Software Productivity Metrics |
production. Although this standard prescribes measurements to characterize the software process, it does not establish software productivity norms, nor does it recommend productivity measurements as a method to evaluate software projects or software developers. This standard does not measure the quality of software. This standard does not claim to improve productivity, only to measure it. The goal of this standard is for a better understanding of the software process, which may lend insight to improving it.

**PURPOSE**
The goal of this standard is to build a foundation to accurately measure software productivity. This is done through a set of precisely defined units of measure. The intention of the standard is to formalize the presentation of productivity data so that it is useful to anyone wishing to improve the software process. This standard is not an end in itself. Instead, it is the beginning of increased precision in collecting and reporting software productivity data. The hope is that this will lead to an improved understanding of the software development process and to improved productivity metrics.


**BRIEF**
The format and contents of software project management plans (SPMP), applicable to any type or size of software project, are described. The elements that should appear in all software project management plans are identified. This standard identifies the elements that should appear in all SPMPs. There are two types of compliance to this standard: format compliance, in which the exact format and contents of this standard are followed, and content compliance, in which the elements of this standard are implemented.

The standard for Software Project Management Plans (SPMP) sections include:

1. Overview
2. References
3. Definitions
4. Elements of the SPMP
   a. Overview (Clause 1)
   b. References (Clause 2)
   c. Definitions (Clause 3)
   d. Project organization (Clause 4)
in a project plan; and *content compliance*, in which the contents of this standard are rearranged in a project plan. In the case of content compliance, a mapping should be provided to map the content-compliant project plan into the various clauses and sub-clauses of this standard.

**PURPOSE**
This standard specifies the format and content of SPMPs. This standard does not specify the exact techniques to be used in developing an SPMP, nor does it provide examples of SPMPs. Each organization using this standard should develop a set of practices and procedures to provide detailed guidance for preparing and updating of SPMPs based on this standard. These practices and procedures should take into account the environmental, organizational, and political factors that influence application of the standard.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e.</td>
<td>Managerial process plans (Clause 5)</td>
</tr>
<tr>
<td>f.</td>
<td>Technical process plans (Clause 6)</td>
</tr>
<tr>
<td>g.</td>
<td>Supporting process plans (Clause 7)</td>
</tr>
<tr>
<td>h.</td>
<td>Additional plans (Clause 8)</td>
</tr>
<tr>
<td>i.</td>
<td>Plan annexes</td>
</tr>
<tr>
<td>j.</td>
<td>Plan index</td>
</tr>
</tbody>
</table>