INTERNATIONAL STANDARD

ISO/IEC/IEEE 24748-3

First edition
2020-10

Systems and software engineering — Life cycle management —

Part 3: Guidelines for the application of ISO/IEC/IEEE 12207 (software life cycle processes)

Ingénierie des systèmes et du logiciel — Gestion du cycle de vie —

Partie 3: Lignes directrices pour l'application de l'ISO/IEC/IEEE 12207 (processus du cycle de vie du logiciel)

Reference number
ISO/IEC/IEEE 24748-3:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Institute of Electrical and Electronics Engineers, Inc
3 Park Avenue, New York
NY 10016-5997, USA

Email: stds.ipr@ieee.org
Website: www.ieee.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by ISO/IEC JTC 1, *Information technology*, SC 7, *Systems and software engineering*, in cooperation with the Systems and Software Engineering Standards Committee of the IEEE Computer Society, under the Partner Standards Development Organization cooperation agreement between ISO and IEEE.

This document cancels and replaces ISO/IEC TR 24748-3:2011, which has been technically revised.

The main changes compared to ISO/IEC TR 24748-3:2011 are as follows:

— revised presentation of concepts, consistent with ISO/IEC/IEEE 12207:2017;

— completely updated presentation of guidance for each life cycle process, including aspects of purpose; outcomes and outputs; activities, tasks, and approaches;

— identified closely related processes;

— identified related international standards for each process, which offer more detailed requirements and guidance.

A list of all parts in the ISO/IEC/IEEE 24748 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

The purpose of this document is to provide guidance on the application of the software life cycle processes standard, ISO/IEC/IEEE 12207:2017. Taken together, the parts of the ISO/IEC/IEEE 24748 series are intended to facilitate the joint usage of the process content of the two high-level life cycle process standards (ISO/IEC/IEEE 12207:2017 and ISO/IEC/IEEE 15288:2015), which in turn may be used together with various more specialized lower-level process standards. In this way, ISO/IEC/IEEE 24748 (all parts) provides unified and consolidated guidance on the life cycle management of systems and software engineering. Its purpose is to help ensure consistency in system concepts and life cycle concepts, models, stages, processes, process application, key points of view, adaptation and use in various domains as the two standards (and others) are used in combination. It should help an organization to design, develop, and sustain software systems using a life cycle model.

ISO/IEC/IEEE 24748-1 provides guidance for the concepts of life cycle management applicable to both systems and software engineering. It covers fundamental concepts such as system-of-interest, stages, processes, projects, and organizations. This document focuses on and expands the coverage of those aspects and processes most relevant to software systems. A companion guidance document, ISO/IEC/IEEE 24748-2, provides similar guidance for the application of ISO/IEC/IEEE 15288:2015.

In conjunction with ISO/IEC/IEEE 24748-1, this document aids in identifying and planning the use of the life cycle processes described in ISO/IEC/IEEE 12207:2017. Since in many respects the Organizational Project Enabling processes and the Technical Management processes are quite similar for software systems to those used for any type of system, this document concentrates on specific guidance for the Technical processes and how they can be effectively used during the software life cycle. ISO/IEC/IEEE 24748-5 focuses on the Technical Management processes, especially Project Planning and Project Assessment and Control, as applied to software projects. The proper use of these processes can contribute to a project being completed successfully, meeting its objectives and requirements for each stage and for the overall project.

This document elaborates on factors, 'best-practice' or typical approaches and methods that should be considered when applying ISO/IEC/IEEE 12207:2017. It does this in the context of the various ways in which ISO/IEC/IEEE 12207:2017 can be applied. It is intended to be useful in a variety of software life cycle situations, including the use of agile methods, which are the most widely used on all types and sizes of projects.

Systems and software engineering — Life cycle management —

Part 3: Guidelines for the application of ISO/IEC/IEEE 12207 (software life cycle processes)

1 Scope

This document is a guideline for the application of ISO/IEC/IEEE 12207:2017. This document establishes guidance to implement a common framework for software life cycle processes, with well-defined terminology, that can be referenced by the software industry. This document provides guidance on defining, controlling, and improving software life cycle processes within an organization or a project. This document recommends methods and approaches suitable for a variety of life cycle models. The guidance emphasizes the importance of establishing a strategy, planning, and the involvement of stakeholders, with the ultimate goal of achieving customer satisfaction.

This document applies to the acquisition, supply, design and development, transition, operation, maintenance, and disposal (whether performed internally or externally to an organization) of software systems, products, and services (including software as a service (SaaS)), and the software portion of any system. Software includes the software portion of firmware. The guidance on processes, activities, and tasks in this document can also be applied during the acquisition of a system that contains software. The guidance in this document can also be used as a basis for selecting, establishing, and improving organizational environments, e.g., methods, procedures, techniques, tools, and trained personnel.

In the context of this document, there is a continuum of human-made systems from those that use little or no software to those in which software is the primary interest. It is rare to encounter a complex system without software, and all software systems require physical system components (hardware) to operate, either as part of the software system-of-interest (SoI) or as an enabling system or infrastructure. Thus, the choice of whether to apply this document for guidance to the software life cycle processes, or ISO/IEC/IEEE 24748-2, depends on the SoI. At a high level, both documents have the same life cycle process framework, but differ in guidance for activities and tasks to perform software engineering or systems engineering, respectively.

The processes and process groups in this document are identical in their purpose and outcomes with those in ISO/IEC/IEEE 12207:2017 and in ISO/IEC/IEEE 15288:2015, with one exception: the System/Software Requirements Definition process of ISO/IEC/IEEE 12207:2017 and this document has a different name from the System Requirements Definition process of ISO/IEC/IEEE 15288:2015.

Use of the guidance in this document is appropriate regardless of software system size or complexity or organizational size. The process outcomes from the ISO/IEC/IEEE 12207:2017 life cycle processes are meant to be generic and applicable to the engineering of any software system in any size organization.

This document does not prescribe nor detail a specific software life cycle model, development methodology, method, modelling approach, or technique and method. The variety of ways for implementing software (e.g., development of new code, integration of existing open source components and commercial products, or modifications to existing software, including transition to new platforms) make it impossible to detail specific procedures.

This document does not establish a management system or provide guidance on the use of any management system standard. However, it is intended to be compatible with the quality management system specified by ISO 9001, the service management system specified by ISO/IEC 20000-1, the

IT asset management system specified by ISO/IEC 19770 (all parts), and the information security management system specified by ISO/IEC 27000.

Clause 6 provides guidance on aspects of purposes, outcomes, activities, and tasks in ISO/IEC/IEEE 12207:2017. However, this document does not repeat the detailed requirements and recommendations for purposes, outcomes, activities, and tasks for each life cycle process found in ISO/IEC/IEEE 12207:2017. Clause 6 also provides references to specialized standards that provide more detailed requirements and guidance for the various processes and information products (information items). This document does not detail information items (process inputs and outputs) in terms of name, format, explicit content and recording media.

NOTE ISO/IEC/IEEE 15289 addresses the content for life cycle process information items (documentation).

2 Normative references

There are no normative references in this document.

3 Terms, definitions, and abbreviated terms

3.1 Terms and definitions

No terms and definitions are listed in this document.

ISO, IEC, and IEEE maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/

— IEC Electropedia: available at http://www.electropedia.org/

— IEEE Standards Dictionary Online: available at: http://dictionary.ieee.org

NOTE For additional terms and definitions in the field of systems and software engineering, see ISO/IEC/IEEE 24765, which is published periodically as a "snapshot" of the SEVOCAB (Systems and software Engineering Vocabulary) database and is publicly accessible at www.computer.org/sevocab.

3.2 Abbreviated terms

API application program interface

CM configuration management

COTS commercial-off-the-shelf

FCA functional configuration audit

IDEF Integration DEFinition

MOE measure of effectiveness

MOP measure of performance

NDI non-developmental item

PCA physical configuration audit

PII personally identifiable information

PRM process reference model

QA	quality assurance
QM	quality management
SaaS	software as a service
SME	subject matter expert
SoI	system-of-interest
SoS	system of systems
TPM	technical performance measure
V&V	validation and verification
VSE	very small entity
WBS	work breakdown structure

4 Concepts for software and software systems

4.1 General

This clause is included to help explain essential concepts as applicable to software and software systems. While understanding concepts does not give the ability to immediately apply them without further thought and work, it is the foundation for their practical use in different project, organizational and life cycle environments.

4.2 Software system concepts

The application of ISO/IEC/IEEE 12207:2017 presupposes an understanding of system concepts. A system is a combination of interacting elements organized to achieve one or more stated purposes. Software is the subsystems or elements of a system consisting of computer programs, related procedures, associated documentation, and data pertaining to the operation of the subsystem or element. Software occurs in most systems, even if it is not the predominant element of interest as it is in a software system.

For the purposes of this document, software systems are considered as created by humans and utilized to provide services in defined environments for the benefit of users and other stakeholders. These systems may be configured with one or more of the following: hardware, software, services, humans, processes (e.g. review process), procedures (e.g. operator instructions), facilities, and naturally occurring entities (e.g. water, organisms, minerals). A system may be considered as a product or as the services it provides. A system element is a member of a set of elements that constitutes a system. A system element is a discrete part of a system that can be implemented to fulfil specified requirements.

NOTE 1 Additional discussion regarding systems, systems of systems, and system structure, is provided in ISO/IEC/IEEE 24748-1. ISO/IEC/IEEE 24748-2 provides more information on concepts related to system life cycle management.

System concepts are directly applicable to software systems. An underlying principle of ISO/IEC/IEEE 12207:2017 is that software engineering applies similar processes to systems engineering, but that software is the leading method for system requirements realization. Consequently, processes are aligned and adapted for methods and approaches relevant for software.

NOTE 2 As applicable to all systems and projects, guidance to concepts is found in ISO/IEC/IEEE 24748-1. ISO/IEC/IEEE 24748-2 includes guidance more specifically applicable to systems where hardware or other non-software elements are the primary concern.

Typically, the system engineering approach to develop a system design is described as a hierarchical, top-down process of systematic decomposition of the system into its subsystems and elements (components). This top-down approach has been traditionally applied to the architecture of software systems as well, from the top-level system-of-interest down through the lowest system element level of the software system structure, such as a line of code. However, the malleable, easily refactored nature of software leads to a different way of considering the software structure. The common practice of producing a minimum of new code and integrating a software system from available components (open source modules, application program interfaces (APIs), software services, support software such as a database management system and a web browser) leads to a more holistic view of its structure. Software systems include a number of components for processing information that can be directly or indirectly related to the software functions and requirements. In compartmentalized software, often none of the sub-systems or components can be considered "top-level" in a hierarchy. Traceability of a high-level stakeholder requirement to each specific subsystem can be difficult.

Characteristic properties at the boundary of a SoI arise from the interactions between subordinate systems. Whatever the boundaries chosen to define the software system, the concepts and models in this document are generic and permit a practitioner to correlate or adapt individual instances of life cycles to its software concepts and principles.

Enabling systems are required for each life cycle process and are typically integrated as an infrastructure for concurrent performance of multiple development, test, and operational processes. Enabling systems are deployed throughout the software life cycle to provide the SoI with support as needed. Each life cycle stage can require one or more enabling systems. An enabling system has its own life cycle; when ISO/IEC/IEEE 12207:2017 (or ISO/IEC/IEEE 15288:2015, if applicable) is applied to it, it then becomes a SoI.

Selection of methods and tools considers software size and complexity, project duration and the number of contributing organizations. Selection of tools should be based on connectivity to other tools that provide inputs or use its output. Ease of use, need for training before successfully using the tool, and availability of administrative support and enabling systems are also factors in tool selection and use. Use of a method or tool does not replace the standard process to be followed but should support the set of activities of a selected process.

4.3 Organizational concepts

An organization is a person or a group of people and facilities with an arrangement of responsibilities, authorities, and relationships. An identified part of an organization (even as small as a single individual) or an identified group of organizations can be regarded as an organization, if it has responsibilities, authorities, and relationships. When an organization, as a whole or a part, enters into a contract, it is a party. Organizations are separate bodies, but the parties may be from the same organization or from separate organizations.

ISO/IEC/IEEE 12207:2017 is intended to be applicable to organizations of all sizes and governance structures, from very small entities (VSE) of fewer than 25 people working on a single project, to large-scale organizations engaged in system of systems (SoS) sustainment for years or decades. The requirements of ISO/IEC/IEEE 12207:2017 are usually stated in terms of "the organization" or "the project," rather than by individual titles or roles. The extent to which tasks and roles can be differentiated depends on the resources available to the organization. Responsibilities that involve checking, validating, or verifying should be assigned to separate persons where the size of the organization permits. In some cases, only one person may be assigned to cover all tasks and take most of the roles and responsibilities.

Modern organizations strive to develop a robust set of life cycle processes that are applied repeatedly to the projects of the organization. To accommodate that need, ISO/IEC/IEEE 12207:2017 is intended to be useful at either the organization level or at the project level. An organization can adopt the standard and supplement it with appropriate policies, procedures, and tools.

A project of the organization typically conforms to the organization's processes rather than conforming directly to the standard. In some cases, projects may be executed by an organization that does not have

an appropriate set of processes applied consistently at an organizational level. Such a project may apply the provisions of the standard directly to a project.

An organization may perform one or more processes as part of its services. A process may be performed by one organization or more than one organization, with one of the organizations being identified as the responsible party. A single project may involve multiple organizations working together as partners. Such a project should use ISO/IEC/IEEE 12207:2017 to establish common terminology, as well as information flows and interfaces among the organizations to enhance communication.

In this document, as in ISO/IEC/IEEE 12207:2017, an organization (or a party) derives its name from the process it is currently performing, for example, it is called an acquirer when it performs the Acquisition process. In ISO/IEC/IEEE 12207:2017, the agreeing parties are called the acquirer and the supplier.

The application of ISO/IEC/IEEE 12207:2017 does not require a specific organizational structure for projects. There are numerous organizational models which can be successfully used to perform the processes of ISO/IEC/IEEE 12207:2017—as long as it is clear who is accountable for providing sufficient resources and for producing each outcome. Organizations can use hierarchical management structures or team-focused, agile work groups. The teams may include representatives specializing in each stage of the life cycle or each life cycle process.

Adaptation and application of ISO/IEC/IEEE 12207:2017 on a project, in which many persons may be legitimately involved, depends on responsibility and accountability. The teams or groups need to be given the appropriate responsibility and authority for doing the work required to meet the project requirements, for example the activities and tasks of a process. A party is identified with overall responsibility and accountability for that entire process, even though the execution of individual tasks may be by different people or groups.

Whether the existing basis for processes is an older version of ISO/IEC 12207 or some other reference point, the fundamental starting point is to identify all the changes needed to go from that basis to ISO/IEC/IEEE 12207:2017. If the existing process basis is an older standard, the amount of changes can be noticeably less than if a different process basis is in use.

NOTE ISO/IEC/IEEE 12207:2017, Annex I provides a mapping from the processes of ISO/IEC 12207:2008 to the processes in ISO/IEC/IEEE 12207:2017. See also ISO/IEC/IEEE 12207-2 for more detailed mappings of outcomes, activities, and tasks.

Whatever the reason for an organization's application of ISO/IEC/IEEE 12207:2017 is, a suggested implementation and transition strategy consists of the following:

a) Identify the desired life cycle model and goals for the organization and the project. Implementing a single process, without considering how it relates to other explicit or de facto processes, is less likely to be beneficial. The Life Cycle Management and Business and Mission Analysis processes can be useful for this effort. If no obvious link is established between this project and the organization's business focus, then lasting commitment to achieve the project goals will be difficult if not impossible to maintain.

b) Plan the implementation. The Transition and Project Planning process can be useful. Identify roles and responsibilities of the project team/organization, assigning a single point of responsibility for each process. In many cases, one individual or organization may be responsible for more than one process, particularly in small projects or organizations.

c) Tailor ISO/IEC/IEEE 12207:2017, if applicable.

d) Conduct pilot project(s).

e) Formalize the approach. A project management plan or change management plan can be useful.

f) Institutionalize the approach, so that the process is supported by top management commitment, organizational policies and procedures, and is used consistently and automatically throughout the project or organization. This also involves measuring performance and implementing process improvement again as necessary.

This strategy is typical of a Change Management approach to introduce changes into an organization or project. The strategy described above may be repeated several times within a project or across an organization as additional processes are implemented or improved.

As with any program which results in changes to work practices, it is essential that the top management within the affected organization is visibly committed to implementing and supporting the changes. In a two-party situation, this can be initiated by a contract and then, as for general organizational use, policies are established with support of top management of both parties.

Bringing all the stakeholders together in this effort is critical: even one area left out that should have been in the planning can materially disrupt applying the new basis. One way of proceeding is for a small group to develop a checklist of transition activities, such as the following:

a) documentation changes, including flow and nomenclature of policies and procedures;

b) staff training needs;

c) responsibility changes, including need for new agreements;

d) impacts on tools and databases; and

e) changes in the inputs required by and outputs from each process.

The initial checklist should then be used by an immediately following, larger, group of all stakeholders to work through what other items need to be added and what the specific changes are for each item on the checklist. Repeated reviews of checklist drafts should be held to find the final few surprises.

Once there is a detailed listing of the changes derived in this or equivalent, manner, the time and cost impact of each needs to be assessed. Then further analysis of the sequence of implementing the changes is necessary. The group should explore phasing in changes in a way that minimizes cost, project disruption and the potential for adverse human reactions. Readiness criteria should be developed for starting each step of the transition, as well as checks for successful completion after each step. Quantitative metrics should be developed and used.

Throughout, a core group should be maintained to oversee the change from one basis to another, with periodic meetings of the entire group of stakeholders.

When a project or organization is already in a steady state, i.e. where the processes have been established and institutionalized, then the implementation strategy can be shortened, considering the risks and opportunities of the project.

4.4 Project concepts

A project is an endeavour with defined scope and start and finish dates, undertaken to work on a product or service in accordance with specified resources and requirements. Typically, a project exists to satisfy an agreement by providing the desired deliverables to the quality expected. For this purpose, ISO/IEC/IEEE 12207:2017 provides a set of Organizational Project-Enabling processes.

Any project is assumed to be conducted within the context of an organization. This is important because a software project is dependent upon various outcomes produced by the business processes of the organization, e.g., employees to staff the project and facilities to house the project.

To perform needed operations and transformations upon software systems during their life cycles, the organization creates and monitors projects. The organization both constrains and supports a project. Examples of such organizational constraints and supports are the following:

— sets standards, policies, and procedures by which projects are carried out within the organization;

— initiates, redirects or terminates projects according to business opportunities and strategies;

— provides requested resources including physical and human within availability and financial constraints;

— provides infrastructure support;

— manages the overall quality of software systems produced by a project for internal or external customers.

Projects have defined scope, resources (including time) and focus. The organization can establish projects that are contiguous with stages in the software life cycle (for example, a project to develop software concepts and requirements, followed by another project to integrate, validate, and verify the software system). Projects can also align with the various organizations responsible for different software components. The scope can involve managing all of the stages of the life cycle, a subset of the stages, one or more defined processes, or one or more process activities. Although any software system should be sustained over its full life cycle, it is common for projects to span only portions of that life cycle.

The time scale of projects can vary, for example, one day or several years. The focus of the project is related to the software and its elements in some form of system structure or stage partitioning. The project places certain demands on the organization and the organization places demands on the project. The project requires physical infrastructure, financial and human resource support to carry out project work.

Relationships can exist between a project and other projects, and subprojects. A subproject is a set of resources and tasks organized to undertake a portion of a project. A subproject may be considered a project by those assigned the work.

Many software "projects" involve continuing delivery of capabilities to sustain a software system. Ongoing sustainment efforts, responsive to changes in the environment of the SoI or to new functional and non-functional requirements, handle a continuous stream of work packages, with the timespan limited only by established budgetary or financial reporting restrictions.

Figure 1 illustrates the concept of a software system and its relationship to an organization and its projects.

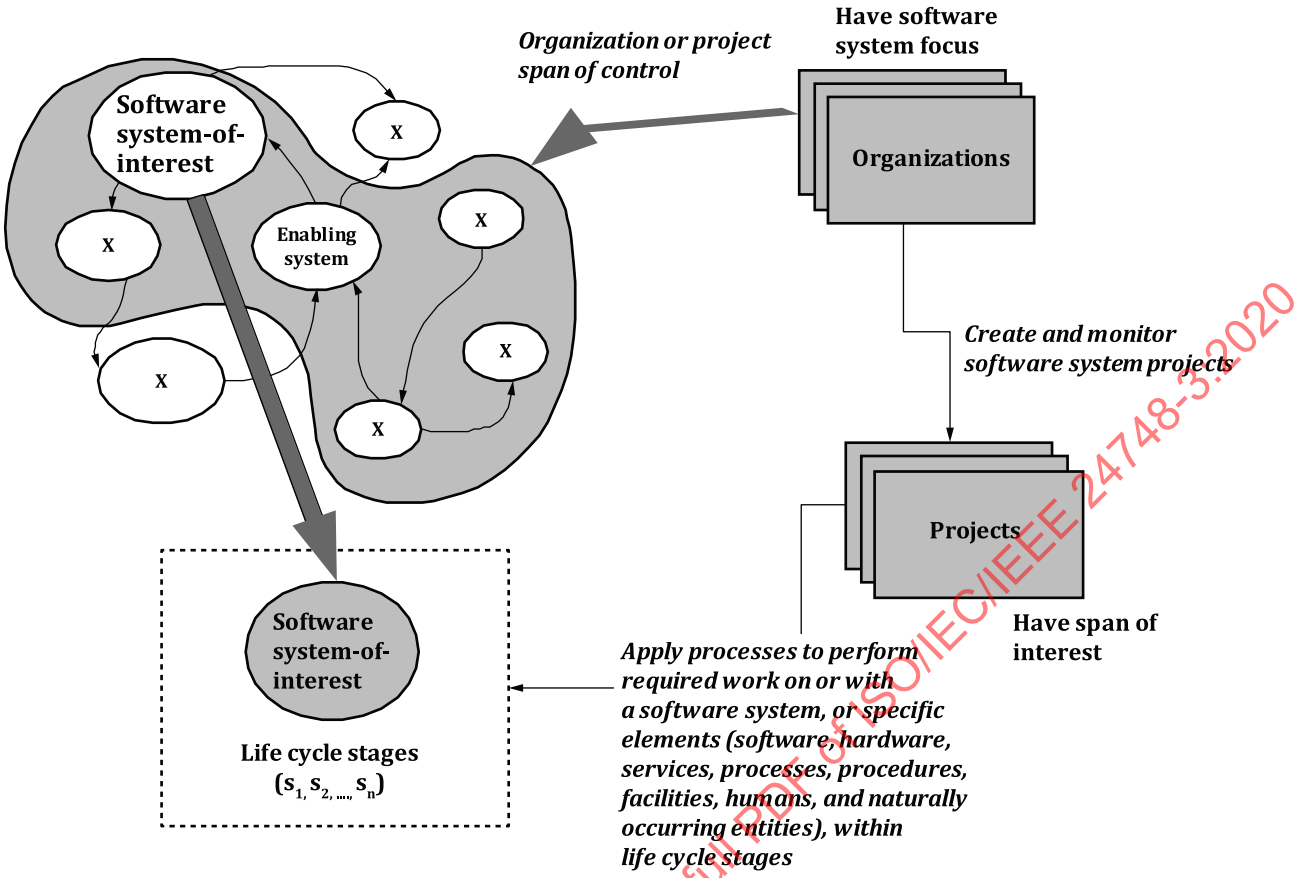

Figure 1 — Organizations, projects, and software systems

Some enabling systems are under direct control of the project. The software and those enabling systems make up the project span of control.

5 Process and life cycle concepts

5.1 Process concepts

A process is an integrated set of activities that transform inputs (for example, a set of data such as requirements) into desired outputs (for example, a set of data describing a desired solution). The software engineering processes in ISO/IEC/IEEE 12207:2017 are defined so that each process has a distinct purpose and outcomes (results). The purpose statement provides the overall rationale for the use of the process. The outcomes are the expected observable results from carrying out the activities of a process. The outcomes provided for each process in ISO/IEC/IEEE 12207:2017 provide a benefit that motivates selection and execution of that process. Attainment of the results fulfills the purpose for performing the process. However, outcomes are not identical with process outputs, which are tangible artifacts and services that can be evaluated. Processes are managed through controls and performed using enabling mechanisms (Figure 2).

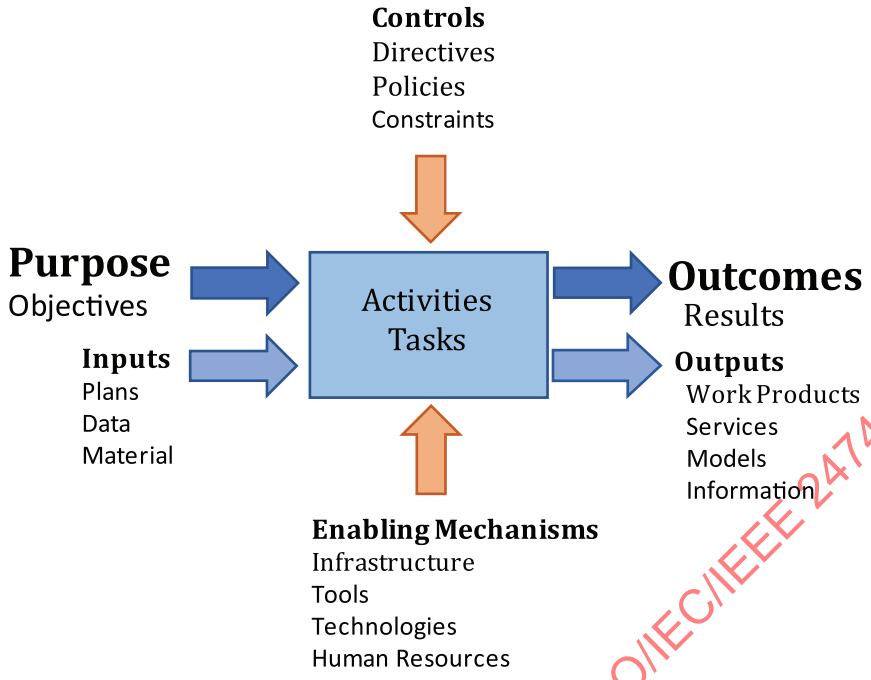

Figure 2 — Process elements

NOTE 1 Additional detail regarding this form of process description can be found in ISO/IEC/IEEE 24774.

Inputs can come from outside an organization or project, or from other processes that precede or accompany the process being examined. Outputs can go to other processes or back to the same process (recursive processing) inside the organization, project (or both), or they can go outside the project or organization, or both. Examples of inputs to and outputs from, a process include the following:

a) information, such as requirements, interface or architecture definitions;

b) data, such as measurements and test reports;

c) material that either ends up in the output or is consumed in producing the output;

d) services that are part of a chain of services, such as setting up a computer prior to, or coincident with establishing an account.

The source of process controls and constraints include organizational or organization management directives and constraints; governmental regulations and laws; project agreements; interfaces with processes used on other systems for which the project is responsible; and internal standard policies and procedures of the organization. Government regulations and laws do not require any specific software design or implementation. They can be viewed as constraining the software system requirements. For consistent application, controls are typically automated through the enabling systems, or applied using methods, procedures, and techniques by the workforce.

ISO/IEC/IEEE 12207:2017 describes the set of processes that are applied to the life cycle of any software system. Therefore, ISO/IEC/IEEE 12207:2017 is designed so that its processes can be applied for a software project of any type, size and complexity, whether focused on tangible products, services, or a mix of both. The processes are designed to be used whether the software is considered as a stand-alone element, or a part of the total software system.

While the processes identified in ISO/IEC/IEEE 12207:2017 are intended to model all the work of software engineering, they do not form a closed loop model. The outputs of one process are not directly identified as inputs for the other processes; the inputs of one process are not always outputs of the other processes. Controls, directives, governance, policy, and organizational strategy and vision drive each process.

Another reason why it is not feasible to create a cohesive, closed-loop model of the software engineering processes is that they are not necessarily sequential. Cohesiveness of a process does not imply that the process can only be performed by one part of an organization at one stage of the software life cycle. Performing a process is not locked to any one time, part of the life cycle, or party. Each process can be applied at any time during the system life cycle and repeated using available inputs at different stages in the life cycle.

The life cycle processes of ISO/IEC/IEEE 12207:2017 can be specialized into process views to reflect specific activities, tasks, and methods appropriate to particular concerns, such as information assurance (security), interfaces, ethics, safety, or various business domains.

NOTE 2 ISO/IEC/IEEE 12207:2017, Annex E presents several examples of process views.

Process performance is accomplished through activities and tasks. An activity is a set of cohesive tasks. A task is a requirement, recommendation, or permissible action, intended to contribute to the achievement of one or more outcomes of a process. A task is expressed in the form of a requirement, self-declaration, recommendation, or permissible action.

The processes, activities, and tasks in ISO/IEC/IEEE 12207:2017 are arranged in a general sequence for ease of presentation. This positional sequence does not dictate the life cycle model sequence. It is intended that the software project select, order, adapt, and iterate the processes, activities, and tasks as applicable or appropriate, using the Life Cycle Model Management process (see 6.2.1).

The processes in ISO/IEC/IEEE 12207:2017 typically follow a similar pattern in the presentation of their activities. It is helpful to be aware of this pattern if an organization wants to combine or subdivide processes or to create new specialty processes. It is more effective when combining tasks from different processes if the tasks relate to the same generic activity. When developing sub-processes, it is better to include at least one activity of each generic type, or to refer to another process where the generic activity is handled for that sub-process.

Table 1 presents the model with examples from a couple of processes.

Table 1 — Model of generic activities within a process

Generic activity	Example activities/task in the Implementation process	Example activities in the Design definition process
Strategize and plan (Plan)	6.4.7.3, a) Prepare for implementation	6.4.5.3, a) Prepare for software system design definition
Perform (Do)	6.4.7.3, b) Perform implementation	6.4.5.3, b) Establish designs related to each software system element.
Evaluate and decide (Check/Act)	6.4.7.3, b) 4) Evaluate software unit and affiliated data or other information according to the implementation strategy and criteria.	6.4.5.3, c) Assess alternatives for obtaining software system elements
Manage outcomes and outputs: Preserve and present artifacts and information items	6.4.7.3, c) Manage results of implementation	6.4.5.3, d) Manage the design.

5.2 Life cycle concepts

5.2.1 Life cycle stages

A software SoI has a life cycle that consists of multiple stages through which the system passes during its lifetime. Within a life cycle stage, processes are performed as required to achieve stated objectives. The progression of a system through its life is the result of actions managed and performed by people in one or more organizations using the processes selected for a life cycle stage. Stages may be interdependent and overlapping, may be of differing durations, and may iterate or be applied recursively.

The size and complexity of a software system affect the work of an organization or project. For example, the tasks performed to accomplish an activity of a system life cycle process or the type and form of work products from application of the processes can be affected. The users of this document are responsible for selecting a life cycle model for the project and mapping the processes, activities, and tasks in ISO/IEC/IEEE 12207:2017 into that model. The organization is responsible for selecting and applying appropriate methodologies, methods, models, and techniques suitable for the project. An organization (for example, an automobile company or medical equipment supplier) or a domain group of an organization (for example, a government defense agency or industry group) often has a unique view of the software system life cycle to control the passage from one life cycle stage to the next. The organization view includes management-focused activities that are used to form both milestones and decision gates. The organization uses these milestones and gates as decision points where investment decisions can be made as to whether a software system should be continued to the next life cycle stage; be modified, cancelled or retired; or be deferred while the plans for the next stage are revised. These milestones and decision gates can be used by organizations to contain the inherent uncertainties and risks associated with costs, schedule, and functionality when a system is created or utilized. To meet the exit criteria of a decision gate, a software system has to be appropriately engineered and the appropriate work products (artifacts and information items) need to be produced to provide decision-making information and required deliverables. Thus, planned engineering activities need to take place during each system life cycle stage to obtain the outcomes and meet the purpose of the stage or a set of stages.

Per ISO/IEC/IEEE 24748-1, the typical system life cycle stages include Concept, Development, Production, Utilization, Support, and Retirement. Use of these terms to define stages is not normative. For software, production of multiple copies is usually not so significant a step as for hardware systems, and other life cycle stages are more common. A common set of stages for a software system is Concept Exploration, Development, Sustainment, and Retirement, with transitions between stages for the system as a whole and for its elements (Figure 3). Another typical set of stages is Conceptualization of a Need, Realization, Utilization, Evolution, and Disposal. A third example of typical stages is Concept, Development, Operation and Maintenance. Another example of software system life cycle stages includes Needs determination, Concept exploration and definition, Demonstration and evaluation, Engineering and development, Production and manufacturing, Deployment and sales, Operations, Maintenance and support, and Retirement.

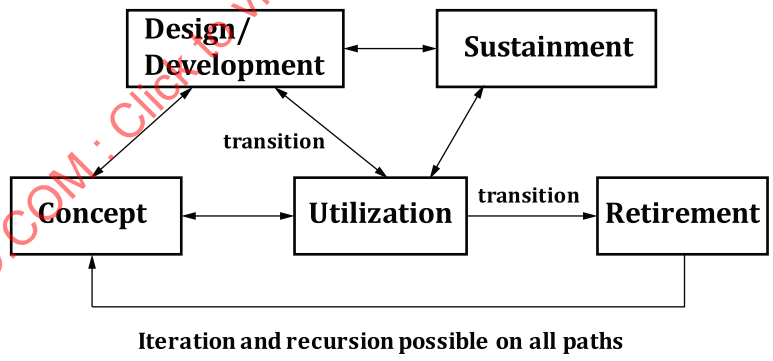

Figure 3 — Typical software system life cycle stages

5.2.2 Interrelationships of software processes and stages

For many reasons, there is not one process per stage. Some processes can be started in early stages for risk mitigation and optimisation of the project. Some complementary processes are performed concurrently to achieve a life cycle stage. With incremental and iterative life cycle models, stages occur several times in the life cycle associated with the same processes.

A manager of a system life cycle stage typically selects the appropriate set of life cycle processes to meet the exit criteria and other stage objectives. For example, the Business or Mission Analysis process, Stakeholder Needs and Requirements Definition process, and System/Software Requirements Definition process are used concurrently and iteratively to develop the concept for a software system. The Architecture and Design Definition processes are used to design the solution for each element

in the system. Application of these processes can be highly iterative in order to arrive at the desired design solution. At the Realization stage, the Implementation, Integration, Transition, Verification, and Validation processes are used in a highly iterative way to produce the software deliverable. During any of the later life cycle stages, a manager can use the Operation process, Maintenance process, and Disposal process to manage the software system while it performs its required functions or is serviced to meet requirements. During earlier life cycle stages, the same processes can be used to help manage the implementation of the system as well as affect the disposal of work products that are no longer needed. The Operation process and Maintenance process should be used to allow the stakeholders to achieve the service or product benefits sought from the system, at the level desired and over the extent of time intended, when they initially defined their stakeholder requirements.

5.2.3 Life cycle process models for software systems

5.2.3.1 General

The life cycle process model of ISO/IEC/IEEE 12207:2017 is intentionally identical to that of ISO/IEC/IEEE 15288:2015 in process names, purposes, and outcomes. To some extent, their activities and tasks are also similarly worded. This approach has advantages in enabling consistent engineering project planning for all kinds of systems. However, this identical model can over-emphasize or under-emphasize some technical aspects of software engineering. The rigorous presentation of numerous processes can be misinterpreted, implying that each process on every project has to be invoked, documented in a written plan, performed, accepted in one or more gated reviews, and results retained in detailed written artifacts. This heavyweight view of the process model has led some software development organizations to avoid the use of the standards, considering them too costly and time-consuming to apply. However, ISO/IEC/IEEE 12207:2017 is intended to be usable in developing process models suitable for all types and sizes of software engineering efforts.

NOTE 1 ISO/IEC/IEEE 24748-7 provides a more rigorous set of activities and tasks for the system life cycle.

Typically, organizations choose sequential, incremental, or evolutionary life cycle models, as described further in 5.2.3.2 to 5.2.3.4.

The processes, activities, and tasks in ISO/IEC/IEEE 12207:2017 are presented in sequence for reference. In some cases, the sequence represents a possible order of performing processes in stages and activities and tasks within processes. However, this positional sequence does not dictate the life cycle model sequence. The organization or software project should select, order, adapt, and iterate the processes, activities, and tasks as applicable or appropriate.

The reasons for applying ISO/IEC/IEEE 12207:2017 processes within an organization can include such situations as the following:

— validating the adequacy of an existing method, e.g., a method that was developed in-house or adopted and extensively modified;

— adapting an existing method to cater for the risks associated with moves into new market sectors where more rigor is required because of perceived risks;

— developing a new method, e.g. to meet the needs of a new organization; this includes organizations created through mergers or business alliances where several process models and procedures already exist;

— managing the introduction of new technology; examples include the automation of existing manual processes, or a change in the methods and tools used to implement a software product;

— evaluating an internal capability of a party to meet agreed criteria, e.g. as part of an acquisition review process;

— establishing a benchmark upon which improvement programs can be developed through an audit.

Implementation of the processes in ISO/IEC/IEEE 12207:2017 can have various scopes:

— use on a single project, either internal to an organization or as part of a two-party contract;

— concentration on some key processes or even a single process where there is expected to be some gain or reduction in risk for an organization;

— adoption across a range of projects in a phased introduction; or

— adoption across all projects and within all parts of the organization; this approach is more feasible for a very small organization.

In practice, some of the life cycle processes, system analysis, measurement, and decision management, are never performed independently but are always an adjunct to another software engineering process. Two processes are often inseparable: it is considered essential to perform software verification (often by testing) at each stage of software development (implementation). In another example, software design, integration, implementation, and validation are usually performed simultaneously and often by the same team members, and the distinction of separately performed processes can be overlooked: all these processes are often considered as parts of a larger stage of software development.

NOTE 2 In ISO/IEC 12207:2008, Validation and Verification activities which are now rigorously separated into distinctive processes were sometimes inserted as part of the same Integration process. That approach reflected that the tasks were closely associated.

The use of DevOps has confirmed the tendency to merge various processes that in the past were more likely to be assigned to distinct teams with separate controls. (DevOps is expanding the work of a software development team and cooperation with the operations team to include management of the software release (Transition) process and the sustainment of software in the operational environment).

Moreover, support for these closely coupled life cycle processes is increasingly automated, and some processes require a significantly reduced level of human intervention compared to practices detailed in previous versions of ISO/IEC 12207 and ISO/IEC TR 24748-3:2011. For example, a human may set up the configuration management (CM) system controls, but how a new day's work is integrated with other branches of the software code being produced that day does not involve human interaction to perform the formal Implementation, Integration, or CM processes of ISO/IEC/IEEE 12207:2017.

The life cycle models for software systems development are not necessarily linked to the software delivery or release schedule. The effort required for transition, the need to utilize software revisions immediately, and the ability of the software system and the customers to handle frequent incremental changes influence whether incrementally produced or evolutionary changes are released frequently or on a regular (e.g., quarterly or annual) schedule.

5.2.3.2 Sequential life cycle model

For new or replacement systems that have extended development cycles (several years) before delivery of the operational system, a sequential approach can be appropriate. The sequential approach has defined decision gates so that an organization can manage an orderly progression of the system from conception through retirement. Many systems, such as those produced for the aircraft or automotive industries, use a similar approach, with development and thorough test before a new automobile model or software system is introduced. Projects using this approach face many challenges, including cost control, funding changes, technology changes, workforce retention, and final customer or acquirer requirements satisfaction. These challenges are created because of the long period from establishing the initial requirements for the embedded software system to the deploying of the system in the marketplace.

This sequential approach can also be applicable to modernization of legacy systems. The software engineering is done on the system being enhanced and its related elements. The impact on the SoI does need to be analysed and where conflicts are revealed, the changes to higher-level systems and the SoI need to be made or the requirements for the applicable system need to be revised.

Because of the long duration of implementation using the sequential approach, several risks should be considered and resolved:

a) expectations and requirements related to the system can change over the years of implementation;

b) knowledgeable workers on teams can turn over;

c) decision making personnel in the organization can change;

d) customer personnel in the acquirer's organization can change;

e) suppliers of system elements and related services can go out of business or change technologies;

f) technical risks can be unresolvable for extended periods;

g) technical obsolescence can arise during a long implementation.

Opportunities can be associated with the sequential approach:

a) the deliberate, stepwise refinement approach whereby the progress of system implementation is carefully evaluated at each milestone allows system quality and risks to be evaluated and investment decisions confirmed before progressing to the next stage of implementation, production lot, or delivery to market;

b) all system capabilities can be delivered at the same time;

c) in-service modification decisions allow determination of whether to do maintenance, a major modification, or to retire the system from service;

d) old systems can be simultaneously retired from service or withdrawn from the market.

In summary, the sequential approach can be effective and efficient for engineering systems where the requirements are well known and stable, deliveries or releases are infrequent, and the system has an extended life span.

5.2.3.3 Incremental life cycle model

The incremental approach applies to organizations that develop (and usually, release) new versions of a product at more frequent planned intervals. The system realized as a result of the concept stage can be a first version. However, it is possible to apply incremental approaches to the software development stage while still using a sequential approach to infrequent software releases.

In this document and in ISO/IEC/IEEE 12207:2017, agile is considered as a method, not as a life cycle process model. Agile can be described as the use of highly iterative methods for the development of requirements and working system increments.

NOTE ISO/IEC/IEEE 12207:2017, Annex H has additional discussion on the application of agile to software engineering. ISO/IEC/IEEE 26515 describes the work of the information developer on an agile team.

Typically, the overall capabilities of the eventual software product are envisaged at the start of system implementation. However, a limited set of capabilities is allocated to the first version. With each successive version, more capabilities are added until the last release fully incorporates the overall capabilities.

Iterative life cycle models involve the operation and support of each version in parallel with the implementation, utilization and support of successive versions. Early versions of the system and support for those versions can be phased out as newer versions are released and used by the customer base or a block modification to earlier versions can be made to incorporate the new capabilities of a later version.

This approach applies mainly to systems that rely on new, enhanced capability versions of the system to be introduced in short intervals so as to remain competitive in the marketplace. Examples include information technology systems such as business systems, medical systems, and routing and firewall

systems. Iterative approaches often begin with an 'Iteration Zero" for such non-deliverable tasks as infrastructure establishment, collecting and prioritizing features to be developed, and developer training.

The incremental approach has associated risks that should be considered and resolved before adopting this approach:

a) Initial versions of the system can have such a limited set of capabilities that customers can be dissatisfied and not be interested in buying the next version.

b) Versions marketed with too short an interval can cause customer dissatisfaction with the cost to upgrade or the retraining costs.

c) Costs for developing and frequently updating training and information for users (time and money) can increase.

d) Expectations may not be met if customers desire the full capabilities in the first version.

e) Poor results may be realized if requirements are not as well understood as originally thought.

f) Unplanned technology changes or competitor system capabilities can require re-direction of the implementation and have a significant impact upon costs and schedule for subsequent versions.

g) The customer may change the requirements as the implementation progresses.

Opportunities can be associated with the incremental approach:

a) Acquirer requirements for early capabilities can be satisfied.

b) The prototypes developed for each early milestone can have a place in the market.

c) Early introduction of the system, even with limited capabilities, can enable exploitation of the marketplace.

d) Consumers and other customers have come to expect frequent improvements in software products, e.g., to respond to security issues, and be wary of products that have not been updated for extended periods.

5.2.3.4 Evolutionary life cycle model

The evolutionary approach also applies to organizations that market new versions of a product at regular or planned intervals. The major difference of this approach with the incremental approach is that the full capabilities of the future versions of the system are not known when an evolutionary implementation is undertaken.

Initially the requirements for the system are partially defined and then refined with each successive version of the system as lessons learned from the use of an early version are translated into new desired capabilities. Often, a new version with enhanced capabilities can replace an earlier version, or a block modification can be made to the earlier version to incorporate the new capabilities of a later version. In this case, implementation of new versions can be done serially or in parallel with partial overlapping. As with versions developed using the incremental approach, different versions can be operated and supported in parallel. Particular care should be taken, however, to maintain configuration information for each version so that users can find operation, training and support procedures appropriate to the version being used.

This approach applies to complex systems for which requirements are not well understood even though the need for the system is understood and approved. These are typically one-of-a-kind or low-quantity production systems. Example systems can include custom information technology systems, military information technology systems, and specific information technology security systems. This approach is also useful for systems that have to achieve quality in use.

During the production stage, either one or a few systems can be produced and delivered, or a large quantity production can be initiated that can continue into the utilization and support stages. The utilization and support stages are typically the longest period of this life cycle and can last for many years. Major systems realized using this approach often have an operation life of decades with modifications using technology refreshments and technology insertions made to sustain the system and lengthen its useful life.

The evolutionary approach has associated risks that should be considered and resolved before adopting this approach:

a) Full capabilities can be preferred at the same time.

b) Training costs can be unacceptable for moving to the next version.

c) There can be uncertainties related to determining future requirements.

d) There can be uncertainties with respect to planning the schedule release of the next version.

e) Configuration control can be a problem.

f) Inappropriate early use of a product prototype in production can cause issues.

Opportunities can be associated with the evolutionary approach:

a) Acquirer requirements for an early capability can be satisfied.

b) Customer feedback can be used to enhance the capabilities of a future version of the system.

c) The prototypes developed to satisfy an early milestone can have a use in the market.

d) Early introduction of a limited capability system can enable countering a competitor threat.

e) Emerging technologies can be exploited.

5.3 Process groups

The software life cycle processes are presented in four process groups: Agreement, Organizational Project-Enabling, Technical Management, and Technical processes. These groups characterize the relationship of the process to the software system and to the software engineering project (Figure 4).

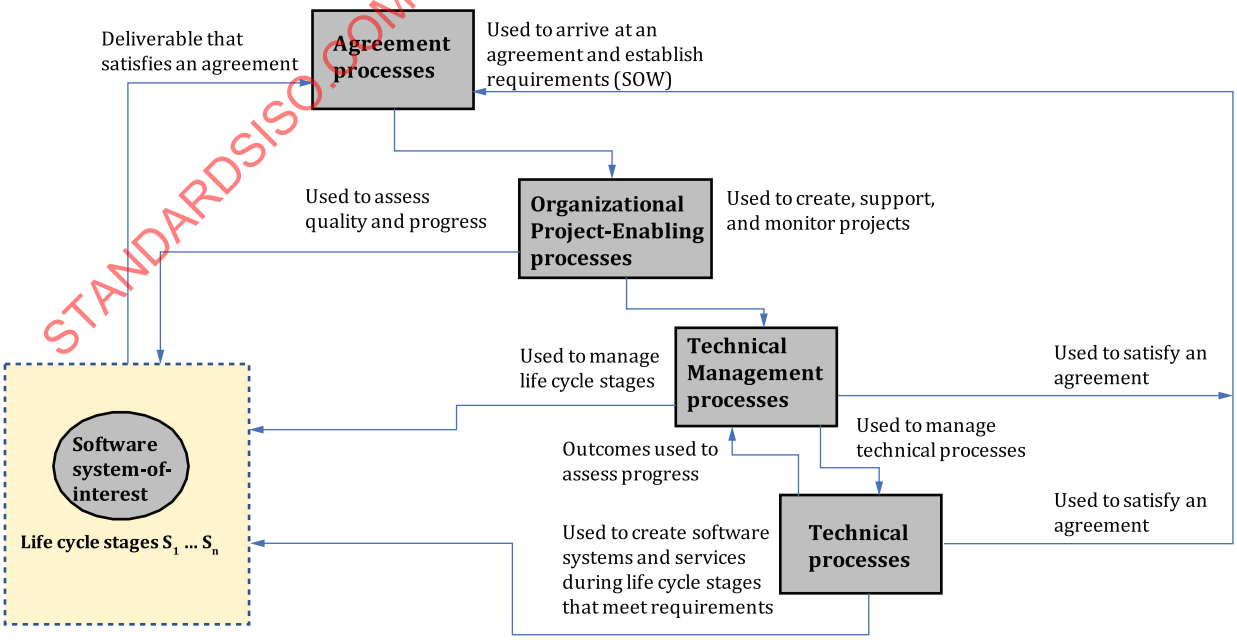

Figure 4 — Relationship of software engineering process groups with the software system

Each of the process groups is involved during the course of a typical project. The project performs the Project Planning process that can consist of updating the plans used to form the response to the acquisition request or the applicable plans from the previous life cycle stage. Appropriate teams are assigned the work required to meet planned requirements. These teams do the work associated with application of the Technical processes to obtain the work products required. The Project Assessment and Control process, Decision Making process, Risk Management process, CM process and Information Management process should be used to monitor, control, and assess the Technical process outcomes to be able to keep work within acceptable cost, schedule, and risk in order to meet performance requirements for the system. The Organizational Project-Enabling processes and Technical Management processes are implemented to provide support to the project, and to review the project, as appropriate.

The Organizational Project-Enabling processes are not assumed to be adequate to operate a business, nor are the Technical Management processes assumed to be sufficient to manage a project. Technical management focuses on managing the technical aspects of the software engineering work, rather than performing general project management. The Organizational Project-Enabling and Technical Management processes, considered as a collection, are intended to state the minimum set of outcomes that the software engineering project requires from the organization. They focus on the tasks and activities related to software engineering, not on the overall governance, business, financial, human resource, or information management policies and procedures of the organization.

Organizations generally use ISO/IEC/IEEE 12207 as part of an effort to improve software products and services through improving their software engineering processes. This may be in conjunction with other process assessment and capability determination methods. As explained in ISO/IEC/IEEE 12207:2017, Annex C, it can be used as a process reference model (PRM) for process assessment. However, neither ISO/IEC/IEEE 12207 nor this document provide a maturity model. All the activities and tasks in ISO/IEC/IEEE 12207 are presumed to be achieved at a level 2 of a typical 5-level maturity model. In practice, many of the activities can be performed more consistently and effectively at a higher maturity level.

NOTE The ISO/IEC 33000 family of standards, particularly ISO/IEC 33004 and ISO/IEC 33020, provide comprehensive guidance on process capability levels, process reference models, and process assessment models.

6 Software life cycle processes

6.1 Agreement processes

6.1.1 Acquisition process

6.1.1.1 Aspects of purpose

The Acquisition process is used to obtain a desired software product or service.

6.1.1.2 Aspects of outcomes and outputs

The acquisition process results in acquisition of all or part of a software system or software services (SaaS), which can be newly developed software, off-the-shelf applications or open source components. The agreement process results in an agreement (formally, a contract), which should include statements of the responsibilities of the acquirer and the supplier, the information and software deliverables which may include software licenses and rights to intellectual property; any warranties, the period of contractual responsibility, and the obligations of the supplier following acquirer acceptance.

6.1.1.3 Related processes

The Acquisition and Supply processes are mutually interdependent; one cannot be executed without the other. However, it is possible for either the acquirer or supplier to execute a formal process conforming to ISO/IEC/IEEE 12207, while the other party uses a less formal or tailored approach.

The Decision Management process and System Analysis process are often used to support trade-offs for the acquisition strategy. Examples include: determining a make or buy decision, as well as the suitability of specific COTS or modified OTS solutions and supplier evaluation.

The Project Assessment and Control process provides input from previous projects to estimate projected cost, schedule, performance, and the impact and risk of undesirable outcomes on the organization.

Acceptance testing can be performed using the Validation process.

The project is opened and closed by the Portfolio Management process.

Changes to system elements or information are controlled through the Change Management activity of the CM or Transition process.

6.1.1.4 Activities, tasks, and approaches

Project relationships are managed through formal or informal agreements in accordance with organizational policies and procedures. Formal agreements can be executed using the Agreement processes of ISO/IEC/IEEE 12207:2017. Depending on the type of project relationship involved, agreements may exist within a single organization, or may span organizational boundaries. Agreements may be between a project and a specific organizational element or elements, among multiple projects, or among a project and its subprojects. Agreements provide a mutual understanding of the problem to be solved, work to be done, established constraints, deliverables, and clearly defined responsibilities and accountability. Agreement on approach is particularly important when the software developer is using methods, such as agile (iterative) methods, that are new to the acquirer or the supplier, as there can be different expectations on stakeholder involvement in the project.

The roles of acquirer and supplier depend on the situation. For example, in a given software implementation project, an acquirer may request a supplier to perform software implementation, with the acquirer and the supplier executing one application of the Agreement processes. The supplier may then request its sub-contractor to perform all or part of the software implementation. The supplier (now in an acquisition mode) and its sub-contractor (in supplier mode) can execute a separate application of the Agreement processes. The acquirer should specify the level of acquirer participation throughout the supplier's project execution to provide the required software product or service.

Typically, in any acquisition situation there are several approaches or ways of doing something. An approach or way that best achieves the overall acquisition goals and constraints is desired. Considerations to include are:

— after-market opportunities and competition;

— business unit policies;

— organization environment;

— applicable legal and regulatory constraints;

— financial resource availability;

— human factors;

— improvement strategy;

— integration and interoperability;

— logistics (supportability);

— obsolescence;

— operational environment;

— producibility;

— ethical considerations

— safety and survivability;

— security;

— stakeholder goals;

— time-to-market constraints;

— potential risks for acquisition and supply.

Each agreement, whether formal or informal, should include the following information to the appropriate level of detail:

a) responsibilities for the work expected to be done, for example in the form of work statements;

b) known functional and performance requirements, attributes, and characteristics that clearly assign what the software system and its related services are expected to do, be like, or contain, including interfaces with other systems, humans, and environments; these can be in the form of a set of formal requirement statements, scenarios or user stories, use cases, or other forms;

c) deliverables, for example products, services, information for use (documentation), and data;

d) entry or exit decision criteria for the life cycle stages.

e) required management and technical reviews to track the fulfilment of the agreement and assess the maturity of the software system;

f) verification and validation responsibilities;

g) acceptance conditions and transition instructions, exception handling procedures, conditions requiring re-negotiation of the agreement, conditions required to terminate the agreement, conditions required to impose penalties or invoke bonuses and payment schedules;

h) other appropriate information, such as:

   1) cost and schedule constraints;

   2) trade-offs between cost, schedule, performance, safety, and product quality level;

   3) implementation delivery milestones;

   4) payment schedules;

   5) rights and restrictions associated with technical data (for example for copyright, intellectual property and patents);

   6) planning documents such as applicable work breakdown structure (WBS), related configuration data, and acquirer supplied engineering plans;

   7) acquirer review of preliminary plans and other specifically identified procedures, practices, and work products should occur during source selection and contract negotiations;

   8) when more than one standard is specified as a basis, a clear order of precedence should be established in order to prevent conflicting interpretations.

The Acquisition process can be initiated with the receipt of an acquisition request such as a formal request for proposal or an informal internal directive for certain work to be done. This can be before a project is formed for a new engineering effort or within a project if it is for project-related work.

One or more methods may be used to monitor supplier's activities. These methods include software measurements, integrated product teams, on-site or remote access to digital data, and joint reviews. The level of formality of the monitoring should be clearly established at a level appropriate to the scope

and context of the agreement, including mutual responsibilities, frequency and mode of monitoring and ways of gauging acceptable execution of the agreement.

Establish who is authorized to accept each distinguished product or service and on what basis, including applicable V&V processes.

6.1.1.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.1.1, ISO/IEC/IEEE 41062, ISO/IEC/IEEE 26512, and ISO/IEC/IEEE 24748-7.

6.1.2 Supply process

6.1.2.1 Aspects of purpose

The Supply process allows a supplier to respond to a request from an acquirer for software systems or services. It includes the provision of agreed deliverables.

6.1.2.2 Aspects of outcomes and outputs

The supply strategy can include custom delivery of acquirer specified software and information products, independent development of software for general commercial or open source distribution, or provision of project-related or ongoing software services in support of any of the Technical Management or Technical processes.

6.1.2.3 Related processes

The Acquisition and Supply processes are mutually interdependent; one cannot be executed without the other. However, it is possible for either the acquirer or supplier to execute a formal process conforming to ISO/IEC/IEEE 12207, while the other party uses a less formal or tailored approach.

The Project Assessment and Control process is used to evaluate the supplied services or products, including the projected cost, schedule, performance, and the impact of undesirable outcomes on the organization.

The change management activity of the CM process is used to control changes to the system elements.

The Business or Mission Analysis process is often used to identify an acquirer who needs a software product or service.

6.1.2.4 Activities, tasks, and approaches

A solicitation is typically from an internal or external business unit (it can be internal to the project). The solicitation does not need to be formal.

Appropriate teams or individuals, depending on the project size and complexity, are assigned to consider and prepare a response to the acquisition request. For smaller projects a single individual may be assigned the responsibility to prepare the response, to do the work, and to create the required work products to be delivered.

The assigned team or individual should perform the activities of the Supply process appropriate to establishing an agreement. First the team or individual should scope a strategy for the response preparation effort and understand the capabilities required to do the requested work. The plan should include a schedule of milestones and decision criteria for submitting a response and consider the goals of the organization or project as well as applicable investment decision criteria.

To determine whether to respond to the request for proposal or to determine the specifics of the response, the Technical processes can be planned and performed to the level of the system structure appropriate to the nature and size of the software system and the software life cycle stage. In

addition, the scope of work, cost of the system, and the feasibility of meeting requirements within given constraints should be determined for various alternatives to meet the requirements. The application of the Technical processes should be in accordance with the plan and should be assessed and controlled using the appropriate Project processes. The Organizational Project-Enabling processes are implemented to the extent necessary to support the technical processes and monitor the outcomes and approve the response, as appropriate.

Considerations to include are:

a) applicable legislation and regulations that apply to the supplier;

b) business unit goals;

c) competition;

d) organization environment and policies and procedures;

e) resource availability;

f) related management, technical and resource risks.

6.1.2.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.1.2, ISO/IEC/IEEE 41062, and ISO/IEC 27036 (all parts).

6.2 Organizational project-enabling processes

6.2.1 Life cycle model management process

6.2.1.1 Aspects of purpose

The establishment of a life cycle model provides a framework in which the processes of ISO/IEC/IEEE 12207:2017 are performed.

6.2.1.2 Aspects of outcomes and outputs

The organization should define and manage potential life cycle policies, processes, models, and procedures to projects via the Life Cycle Model Management process.

6.2.1.3 Related processes

The life cycle model management process frequently involves use of the Tailoring process (see Annex A).

The life cycle model influences Project Planning.

6.2.1.4 Activities, tasks, and approaches

The three main activities of Life cycle model management process are the following

a) the establishment of process models and standard processes and procedures for the organization;

b) audits to assess the level of process execution and the adequacy of the models;

c) process improvement.

The selection, implementation and use of a life cycle model by an organization depend on several factors:

a) the acquisition policy of the organization;

b) the nature and complexity of the software system;

c) the stability of system requirements;

d) technological opportunities;

e) the need for different system capabilities at different times;

f) the availability of resources.

The software life cycle model should be constructed considering the desired outcomes of the supply, implementation, operation, maintenance, and applicable supporting and organizational processes, even if a project will primarily focus on one process or stage. Standardization of life cycle processes within a business unit can vary. However, organizations typically encourage all projects and functional business units to use common practices and standards. selection of standards, methods, tools, and computer languages may include selection of the implementer's own standard practices,

New project processes can be formed or an activity under one of the system life cycle processes can be elevated to the process level. For example, selected activities can be included as part of the definition of other processes, an activity under one of the system life cycle processes can be elevated to the process level. or an entire process could be subordinated to an activity level under another process.

6.2.1.5 Related standard*s*

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.2.1; guidance on the development of the life cycle model is given in ISO/IEC/IEEE 24748-1.

6.2.2 Infrastructure Management process

6.2.2.1 Aspects of purpose

Infrastructure management provides and maintains the facilities, tools, hardware, software, services, standards, and tools used to support organizations and projects engaged in the software life cycle. When implemented as a system, the infrastructure elements form enabling systems, such as software development and test environments for target platforms. Enabling systems can also interact with the SoI in the operational environment.

NOTE ISO/IEC/IEEE 12207:2017 uses infrastructure in a general sense for the entire hardware and software environment. Enabling system is used for a specific supporting system.

6.2.2.2 Aspects of outcomes and outputs

The outcome of infrastructure management is an efficient environment to support the SoI. Frequently, integrated enabling systems and tools which reduce the labor of manual data entry and data checking are considered more efficient. To reduce the risk of transition, infrastructure systems for software development and testing that duplicate or offer a scaled-down version of the operational environment are preferred.

6.2.2.3 Related processes

In ISO/IEC/IEEE 12207:2017, almost every Technical process has a task to obtain the appropriate infrastructure and enabling systems, and to validate their fitness for use. These tasks are inserted into the other processes, rather than presented only once in the Infrastructure Management process, so that the need for enabling systems is not overlooked by the responsible parties.

The Operations and Maintenance processes are applied to manage the infrastructure. All of the technical management and technical processes can be applied to infrastructure systems, just as they are applied to deliverable software products.

6.2.2.4 Activities, tasks, and approaches

An infrastructure of integrated enabling systems should apply to all parts of the life cycle and potentially support each process. For each architectural design solution in the system structure the enabling system requirements related to the system should be identified. Timely availability of enabling systems is critical to productive use of human resources, software quality, and keeping the work on schedule. Some or possibly all enabling systems can be outside the direct responsibility (boundary) of the project. Some or all of the enabling systems can already exist within the project's organization. Other enabling systems can be easily made available, for example by rental or purchase. However, one or several enabling systems may not exist, and have to be created and made available in time to provide required services. For example, test software may need to be developed or adapted along with the development of the application system-.

The infrastructure is typically specified through appropriate policies and standards and includes identification and dissemination of resources (staff, schedule, budget), requirements (contractual, standards), and expected outcomes (cost, benefit). Typically, infrastructure is established based on: organizational strategic plan, capabilities, resources, risk levels, value to the customer, and technology policies. For a business, payoff goals, market segment, market position, investment and competitive advantage are also factors.

The organization should establish, use, and continually refine metrics that show how well the infrastructure is supporting the needs of the organization in executing its mission.

Project planning and organizational policy should define what software environments will be made available to each project or for use throughout the organization. At a minimum, developers should not create software modifications in the live production environment. Separate environments for the enabling systems are common for a variety of purposes:

a) 'sandbox' installation and testing of new versions of enabling systems;

b) software development, including different environments for long-term upgrades and releases and immediate fixes;

c) software test (multiple environments possible for concurrent unit tests, regression tests, and qualification or acceptance test);

d) software training;

e) production operations.

An important factor in establishing software environments is selection of the application data to be available there during transitions and data migration testing. Often the size of the database or the application data is limited in non-production environments for cost considerations. To some extent performance modelling and simulation can compensate for smaller data loads in performance testing. However, estimating durations for major data migrations can entail performing dry runs of the process for a full database backup and data load. Also, when the software system uses personally identifiable information (PII), the organization needs to determine if copies of that live data will also be used in development, testing, documentation, and training. Where possible, it is preferable to use fictitious or anonymized data to avoid unintended disclosure of PII.

6.2.2.5 Related standard

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.2.2.

6.2.3 Portfolio Management process

6.2.3.1 Aspects of purpose

A project portfolio is a collection of projects that addresses the strategic objectives of the organization. The portfolio management process leads to an investment decision and the authorization, initiation, and closure of projects, as well as development and termination of services, products, and product lines.

6.2.3.2 Aspects of outcomes and outputs

The outcomes include a portfolio of projects started or closed, and the creation and ending of product line (product family) or software system offerings.

6.2.3.3 Related processes

The Decision Management, Measurement, and System Analysis processes support the comparative analysis of alternatives.

The Risk Management process is used to identify and evaluate opportunities (sometimes called positive risk) as well as adverse (negative) risks in the portfolio.

The Project Planning process is used to carry out decisions regarding the product portfolio or service catalog.

Market surveys performed during Business or Mission Analysis are applied to Portfolio Management.

6.2.3.4 Activities, tasks, and approaches

Appropriate management aids are typically established to enable availability of valid information for directing and enabling projects, including project status, standardized automated tools, and organizational policies, procedures, and products available for reuse.

Portfolio management considers the interrelationships and dependencies among projects in making decisions.

Technology assessment should be conducted continually for innovations that could improve products in the portfolio or result in obsolescence or impacts on the competitiveness of the existing portfolio.

6.2.3.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.2.3, and the ISO/IEC 2655x family of standards.

6.2.4 Human Resource Management process

6.2.4.1 Aspects of purpose

The Human Resource Management process is used to identify, train, and make available people with appropriate skills, education, and experience for roles on software engineering projects. This process includes both employees and contractors.

6.2.4.2 Aspects of outcomes and outputs

The outcome of Human Resource Management is an organization and projects staffed with capable, competent, and productive individuals. The establishment of human resource services enables organizations to achieve their goals and objectives within legal, financial and other constraints and agreement requirements.

6.2.4.3 Related processes

Knowledge Management involves the capture of knowledge held by experts so that it can be applied by others.

6.2.4.4 Activities, tasks, and approaches

Team or group members assigned responsibility for a software system need to be available and competent. Typical activities of Human Resource Management include resource acquisition, skill development through training or coaching, skill deployment to projects and organizational needs, and skill measurement and assessment. In the context of complex software systems, skill deployment can involve multidisciplinary teams or groups.

As a rule of thumb, project teams consist of seven plus or minus two members to develop the teamwork necessary for maximizing efficiency and effectiveness. The scope of the project, organizational model, and life cycle model determine the number and types of teams needed at any stage. Typically, the team needs to rely on the individual specialist or functional groups of the organization to do such tasks as assessments on security, safety, survivability, reliability, and effectiveness as well as trade-off studies, risk analyses and design work. In this context, the team then becomes the integrated decision-making structure for process activities performed in a software life cycle stage. It is, however, important that individual teams share knowledge and communicate with other teams working on enabling systems for the same system and other adjoining systems. Such communications should be established so that the resultant software is properly integrated from the lowest level up. It is also important so that each system and system element in the system structure is appropriately supported over its life.

6.2.4.5 Related standard

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.2.4.

6.2.5 Quality Management process

6.2.5.1 Aspects of purpose

Quality Management (QM) provides a sufficient level of confidence that system and service quality characteristics for each project are adequately defined and activities are effectively and efficiently managed so that customer requirements are met and other parties interested in organizational success are satisfied.

6.2.5.2 Aspects of outcomes and outputs

QM should be responsible for establishing organizational policies and resources for ongoing evaluations of software acquisition, supply, implementation, maintenance, operation, and supporting process activities and the resulting software products, as applicable, to

a) assure that each process, activity, and task required by the contract or described in plans is being performed in accordance with the contract and with those plans, and

b) assure that each software product required by a relevant process or contract provisions exists and has undergone software product evaluations, testing, and problem resolution, as required.

6.2.5.3 Related processes

Quality Assurance (QA) performs, witnesses, or monitors the detailed evaluations of products and services that are established and supported by QM. Often QM is at the organizational level and QA operates at the project level.

6.2.5.4 Activities, tasks, and approaches

While every member of the organization and project team should be responsible for monitoring and upholding the quality of products and services, it is desirable to maintain organizational independence for QM and QA. Quality work should not be under the direct operational control of the project or product line manager and should report independently to a higher level of management. This approach builds confidence that the reports and recommendations from QM and QA are not suppressed due to expediency on the project. QM and QA should have the organizational freedom, resources, and authority to plan for objective evaluations.

6.2.5.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.2.5, IEEE 730, ISO/IEC 25010, ISO/IEC 25030, ISO 9000, ISO 9001, ISO 10007, and ISO/IEC/IEEE 90003.

NOTE ISO/IEC/IEEE 12207:2017 does not establish a quality management system, but its activities are consistent with ISO 9001.

6.2.6 Knowledge Management process

6.2.6.1 Aspects of purpose

Knowledge management retains and shares subject matter expertise relating to business and technical domains, software products, and procedures.

6.2.6.2 Aspects of outcomes and outputs

Knowledge management is particularly applicable to the expertise of individuals who developed, operated, or maintained legacy product lines, software systems, and customized processes, where written documentation is inadequate or non-existent. Without knowledge management, familiarity with system configurations, emergency procedures, software internals, and maintenance processes can be lost when the subject matter expert (SME) is no longer available. A desirable outcome of knowledge management is a documented knowledge base that can be utilized to train and guide new developers, maintainers, and operators of the software system or for use in transition to a replacement system.

6.2.6.3 Related processes

Knowledge management is allied with CM of systems and components and Information Management of documented procedures and design artifacts.

Human Resources can assist with locating SMEs.

6.2.6.4 Activities, tasks, and approaches

Records from knowledge capture activities may be in various forms, including interviews, recorded demonstrations, and written procedures. Knowledge can be documented in many forms, such as written presentations, videos, podcasts, and application help.

Knowledge management can also involve transfer of information about obsolete technology to maintained content management systems.

NOTE 1 Reuse of software assets is related to knowledge management. It would be exceedingly rare to develop new software without reuse of any existing software elements. Reuse is often accomplished by use of open source libraries. When re-using software from open source, it is important to consider the reliability and potential security risks of software elements from unknown contributors.

NOTE 2 Reuse of software with variations throughout a product line is another approach for reducing risk and improving productivity in software architecture, design, development, and testing. Product line engineering can be considered a form of domain engineering, in which a family of systems is developed within the domain, including their commonalities and variabilities.

6.2.6.5 Related standard

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.2.6. The product line standards in the ISO/IEC 2656x family numbered from 26550 through 26599 detail that approach for numerous software engineering technical and technical management processes.

6.3 Technical Management processes

6.3.1 Project Planning process

6.3.1.1 Aspects of purpose

A project plan is often used as a basis of an agreement between projects and various organizational elements. Planning enables a process, service, or project to be efficiently and effectively accomplished.

6.3.1.2 Aspects of outcomes and outputs

ISO/IEC/IEEE 12207:2017 often refers to planning (an activity) rather than requiring plans (documented output of planning). Common outputs of planning are WBS and plans of various degrees of formality and detail, schedules, and budgets. Software plans for a system, service, or project are often captured in a Software Development Plan (SDP), Software Engineering Management Plan, or Software Engineering Plan. Plans include the scope, tasks, methods, tools, measures, risks, and resources to perform applicable technical processes with the support of applicable Technical Management processes. The plan is consistent with organizational goals and stakeholder requirements.

Event-based scheduling is used to establish events (for example, milestones), significant activities, and tasks related to an event, and the criteria by which completion of activities or tasks are determined.

6.3.1.3 Related processes

The context and authorization for project planning are provided through the Portfolio Management process.

Project Assessment and Control provides input for replanning based on project status.

6.3.1.4 Activities, tasks, and approaches

The formality and level of detail of Planning should be adapted to suit project or work complexity, uncertainty, risk, and resources including funding. Informal small projects may proceed without a formally documented plan, relying on the organization's overall policies and procedures for performing the work. In larger and more complex projects, it is recommended to have a project plan, WBS (or agreed output for near-term increments) and list of deliverables. Once agreed, project plans and budgets should be under change control. Planning may reference, rather than include, specified standards, methods, tools, practices, and computer programming languages.

Planning typically is based on discrete estimates to deliver work packages in a WBS which models the system structure and applicable technical management activities. In agile projects, planning is based on user stories or features which are assigned to iterations (often called sprints). Required milestones, such as technical reviews or product deliverables, influence start dates and resource requirements. Planning includes technical activities and tasks; support tasks to acquire and deploy resources; and management tasks to direct, monitor, and approve the technical and support work. In developing and documenting project management plans, the organization should consider support to be provided by suppliers, such as the implementer, operator, maintainer, as applicable. In preparation for project assessment, project planning should consider the need for independent V&V, joint reviews, audits, and communication methods between the supplier and the acquirer.

Replanning typically occurs before a new iteration, project phase or stage, or as required to adjust for variations and anomalies in process outcomes. Contingency options are used in a plan when there

are known risks or opportunities (for example, significant changes in budget, schedule, requirements, technology, or resource availability) that can cause the project to be redirected.

6.3.1.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.3.1. Planning for the project is often captured in a Project Management Plan. ISO/IEC/IEEE 16326 provides more detail on project planning. ISO/IEC 24748-4 describes systems engineering planning. ISO/IEC 24748-5 describes software development planning and provides an annotated outline for an SDP.

6.3.2 Project assessment and control process

6.3.2.1 Aspects of purpose

Project assessment determines the continuing consistency and relevance of project plans, as well as technical progress using defined technical metrics based on estimated achievement and milestone completion. Project assessment also determines technical variances from project estimates, including variances to cost, availability, and performance specifications. Project control aids the capture and management of outcomes from project management and technical work including the redirection of that work to overcome obstacles, to respond to changing circumstances or to correct variances.

Technical management reviews have the following purposes:

a) keep management informed about project status, directions being taken, technical agreements reached, and overall status of evolving software products;

b) resolve issues that could not be resolved during technical reviews;

c) arrive at agreed-upon mitigation strategies for near-term and long-term risks that could not be resolved at technical reviews;

d) identify and resolve management-level issues and risks not raised at technical reviews;

e) obtain commitments and acquirer approvals needed for timely accomplishment of the project.

Technical reviews may have the following objectives:

a) review evolving software products that have met their completion criteria; review and demonstrate proposed technical solutions; provide insight and obtain feedback on the technical effort; and surface and resolve technical issues;

b) review project status and identify near-term and long-term risks regarding technical, cost, and schedule issues;

c) arrive at agreed-upon mitigation strategies for identified risks, within the authority of those present;

d) identify risks and issues to be raised at management reviews;

e) ensure on-going communication between technical personnel of the acquirer and supplier, or of the project and related support processes (QA, CM, V&V) and user representatives.

NOTE Technical reviews validate that deliverables have the expected quality and that the project will meet its technical objectives.

6.3.2.2 Aspects of outcomes and outputs

Project assessment determines whether to proceed to a new stage or continue work to satisfy the entry/exit criteria of a specific life cycle model. To control and correct technical variations between expected results and assessment results, results are analyzed to detect trends and identify root causes of problems.

Measures are defined and used to collect data to assessment of customer satisfaction.

Project control results in recommendations for corrective action, based on trade-off and causal analyses and analyses of resulting impacts on cost, schedule, performance, and risk. When variations are significant or cannot be corrected by re-performing the process tasks that generated the outcome data, the project planning process is re-initiated in order to plan and implement appropriate corrective actions. One result of replanning can be a decision to reduce project scope, to avoid increases in cost or schedule or to defer achievement of performance requirements.

Project control can also take advantage of results that are better than planned, by accelerating or taking on extra work, reducing resource requirements, and cutting costs.

6.3.2.3 Related processes

Project Planning establishes the framework for Project Assessment and Control.

General process assessment is performed through the Quality Management process.

The Measurement process is essential for project assessment.

6.3.2.4 Activities, tasks, and approaches

Assessment can be applied to the technical, schedule, or cost performance outputs of all the technical and technical management processes, as well as to the technical infrastructure and services. Project assessment considers progress and achievement against plans (work productivity) and against technical requirements (product quality). It evaluates adherence to practices and procedures.

In iterative life cycle models, teams hold a “Retrospective” review of what went well during the iteration, what did not go well and what the team can do to improve its processes, including improvements to individual and team practices, processes or tools.

Earned value methods can be used to determine if the expenditure of time and funding is consistent with plans and with work achieved. For example, the budgeted cost of the work performed is compared to the budgeted cost of the work scheduled and the actual costs of work performed. These measures are then used to determine estimates at completion, estimates to complete, and cost and schedule variances.

Measures are defined and used to collect data on customer satisfaction. This can be done based on follow-up to customer support calls, customer satisfaction surveys, or acquirer evaluations of service.

Reviews are an important mechanism for project control. reviews should be attended by persons with authority to make technical, cost, and schedule decisions at an appropriate level. Typical review objectives include determination of:

a) system maturity and how well the solution satisfies requirements;

b) traceability of requirements, the validity of assumptions and decision rationale;

c) identification of un-resolved issues and issues not determined during project work;

d) related risks, needed resources, and adequacy of preparation for conducting the next system life cycle stage.

Technical reviews should be conducted for each project as appropriate to the engineering view used, such as the following:

a) a review held prior to performing the Requirements Analysis process to help ensure that the stakeholder requirements are complete, consistent with the acquirer’s intent, understood by the supplier and have been validated; this review can prevent proceeding with a less than acceptable set of requirements;

b) a review conducted to consider all concepts looked at and to determine whether the preferred concept has the potential of satisfying defined stakeholder requirements and is based on a set of viable, traceable technical requirements that are balanced with respect to cost, schedule, and risk;

c) an evaluation of the established requirements baseline to balance the set of technical requirements with respect to cost, schedule and risk;

d) an evaluation of the established functional baseline to ensure that the system definition is based on achievement of technical requirements; it also can be used to ensure readiness to proceed with the preliminary design of each system of the system structure;

e) a review conducted for the preliminary design of each system of the system structure to confirm that:

   1) the specifications and other configuration descriptions are defined appropriately;

   2) the design solution is consistent with its acquirer's requirements;

   3) enabling system requirements are sufficiently defined to initiate enabling system implementations, as required, or to acquire the applicable enabling systems;

   4) approaches planned for developing designs for pre-production prototypes are appropriately planned;

   5) risks are identified and resolution plans are feasible and judged to be effective;

f) a review conducted for the detailed design of each system of the system structure to demonstrate that:

   1) software descriptions, such as scenarios, user stories, use cases, models, prototypes, specifications, and diagrams, are sufficiently defined to realize the design solution through implementation or integration, as appropriate;

   2) the design solution is consistent with its acquirer requirements;

   3) enabling system requirements to provide life cycle support have been adequately defined to initiate enabling system implementation or acquisition, as appropriate.

g) reviews conducted prior to each scheduled series of tests on an implemented or integrated test system to ensure test readiness by confirming that all test related enabling systems are in place and the test environment is prepared to accomplish test objectives;

h) reviews conducted prior to releasing each design solution for first system or batch production to ensure production readiness by confirming that production enabling systems and materials are in place and the production environment is prepared to accomplish production objectives.

After completion of the detailed design of each system in the system structure that is based on the allocated baseline, and with proof that the production system is ready and other enabling systems are ready or are expected to be available when needed, the system can be released for production. The system produced can be a one of a kind, the first of a limited version or the first of many that will be produced.

Risk items should be included in reviews.

In addition to contractually required reviews, the supplier, including the implementer, maintainer, or operator, as applicable, may propose additional joint management reviews. The supplier and other applicable parties should plan and take part in such additional reviews at locations and dates proposed by the supplier and approved by the acquirer. These reviews should be attended by persons with authority to make cost and schedule decisions and may have the following objectives:

a) keep management informed about project status, directions being taken, technical agreements reached, and overall status of evolving software products;

b) resolve issues that could not be resolved at joint technical reviews;

c) arrive at agreed-upon mitigation strategies for near-term and long-term risks that could not be resolved at joint technical reviews;

d) identify and resolve management-level issues and risks not raised at joint technical reviews;

Obtain commitments and acquirer approvals needed for timely accomplishment of the project. The supplier, including the implementer and/or the maintainer and/or the operator, as applicable, should plan and take part in joint technical reviews at locations and dates proposed by the supplier and approved by the acquirer. These reviews should be attended by persons with technical knowledge of the software products to be reviewed. Support process disciplines (e.g., QA, CM, V&V) should provide input to or be present at joint reviews. The reviews should focus on in-process and final software products, rather than materials generated especially for the review. The reviews may have the following objectives:

a) review evolving software products that have met their completion criteria; review and demonstrate proposed technical solutions; provide insight and obtain feedback on the technical effort; and surface and resolve technical issues;

b) review project status and surface near-term and long-term risks regarding technical, cost, and schedule issues;

c) arrive at agreed-upon mitigation strategies for identified risks, within the authority of those present;

d) identify risks and issues to be raised at joint management reviews;

e) ensure on-going communication between acquirer and supplier technical personnel.

The level of formality of the monitoring should be clearly established at a level appropriate to the scope and context of the agreement, including mutual responsibilities, frequency and mode of monitoring and ways of gauging acceptable execution of the agreement.

6.3.2.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.3.2, ISO/IEC/IEEE 16326, ISO/IEC/IEE 24748-5, ISO/IEC/IEE 24748-4, ISO/IEC/IEEE 24748-8, and ISO/IEC/IEEE 15289.

6.3.3 Decision Management process

6.3.3.1 Aspects of purpose

The Decision Management process is used to select among alternatives based on organizational policy, evidence, and experience.

6.3.3.2 Aspects of outcomes and outputs

The outcomes of the Decision Management process are methods and tools for obtaining and evaluating evidence. The primary outputs are recorded decisions.

6.3.3.3 Related processes

Decision management supports all processes on various levels.

6.3.3.4 Activities, tasks, and approaches

Rationale for key decisions should be documented. The rationale should include analysis methods, criteria used to make those decisions, and trade-offs considered. What constitutes “key decisions” in the selected life cycle and the approach for providing the rationale should be described in the organizational policies. Key decisions typically include whether or not to proceed with a project,

whether to advance from one stage to the next, whether to expend substantial resources beyond those planned, and whether the software is safe and ready for use.

During the software life cycle stages, management can use established decision gates to determine whether the objectives (financial as well as technical) were satisfactorily completed and whether the system is ready to progress to the next stage.

The concept stage of a software system life cycle model typically has two major decision gates. The first decision gate in the concept stage is after a period of preliminary analysis. Prior to this decision gate appropriate research and implementation is carried out, technology challenges and opportunities are explored, and potential system concepts are analysed. Those concepts that have promise for future business opportunities are presented to management for approval to continue the implementation of the more promising concepts. The concepts can be needed to develop a new market, to respond to a specific threat or to respond to a request for proposal.

A second decision gate typically occurs after studying the feasibility of the alternative concepts. The organization should determine whether to proceed with a selected course of action:

a) whether a concept is feasible and is considered able to counter an identified threat or exploit an opportunity;

b) whether a concept is sufficiently mature to warrant continued implementation of a new product or line of products;

c) whether to respond to a request for proposal (supplier decision) or to approve a proposal (acquirer decision)

Decision gates provide the opportunity for management to review the plans for execution before giving the go-ahead. Typically, there are decision gates associated with initiation of stages in the software life cycle, such as beginning to define the architecture and design or software development. Another major decision gate is before initiating transition to operation (production).

In order to meet the exit criteria of a decision gate, the software system should be appropriately engineered. The appropriate work products need to be produced to provide decision-making information and required deliverables. Thus, planned engineering activities need to take place during each system life cycle stage to obtain the outcomes and meet the purpose of the stage or a set of stages.

During later life cycle stages while the system is being used (utilization and support), the decision gate is typically used to make three kinds of decisions:

a) whether system technology should be refreshed (change to the baseline configuration with or without changing functions and performance);

b) whether new system technology should be inserted (change to system performance);

c) whether the system should be retired.

6.3.3.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.3.3 and IEEE 7000 (in preparation).

6.3.4 Risk Management process

6.3.4.1 Aspects of purpose

Risk Management reduces or eliminates the possibility and impact of potential adverse events affecting a project. Considered in a positive sense, Risk Management allows an organization to take advantage of opportunities it discovers.

6.3.4.2 Aspects of outcomes and outputs

Risks are considered resolved when the possible consequences are acceptable. Acceptable means that the project and affected stakeholders can live with the worst-case outcome.

6.3.4.3 Related processes

Risk Management is inherent in Project Assessment and Control. Risk Management should be considered when applying any process.

6.3.4.4 Activities, tasks, and approaches

Traditionally, risk management includes the following:

a) risk planning for the risk management cycle;

b) risk identification;

c) risk assessment to characterize and categorize the risk, including identifying sources of risk, the likelihood of the risk, and potential effects and their impact;

d) risk control to resolve prioritized risks by developing risk resolution actions, establishing triggers for implementation of risk resolution actions, monitoring risk status, implementing risk resolution actions and correcting deviations.

Risk assessment is based on experience or awareness of others' experience to identify likely risks, estimate their probability and worst-case impact, identify mitigations, and evaluate the impact of the mitigated risk. More detailed risk analysis can involve quantifying the uncertainties and assumptions affecting the risk analysis.

For software projects, risks can be categorized by their impact as project risk, process risk, and product risk:

a) project risk – organizational, operational, or contractual concerns including resource constraints, external interfaces, supplier relationships, contractual restrictions, lack of organizational support, and vendor unresponsiveness;

b) process risk – degree of experience with the process and previous successful use of the process; process risk is more extreme with a new team or changes in established procedures, such as when a large team begins using agile methods without training or pilot projects.

c) product risk – for software, degree of familiarity with the business domain, the development tools and software languages, requirements stability, complexity of the business processes and interfaces, and feasibility of testing the software performance are sources of product risk.

NOTE Risks can also be categorized by their source (the type of threat), e.g., external or internal risks: or security, safety, and ethical risks.

Risk resolution includes risk acceptance, risk avoidance (not taking actions that could incur the risk), risk sharing or risk transfer (insurance, contingency funding, resilient or redundant systems), and proactive risk mitigation actions that reduce the probability or the severity if the risk becomes a real situation. Risk resolution actions are prioritized according to risk impact and available resources.

6.3.4.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.3.4, and ISO/IEC/IEEE 16085.

6.3.5 Configuration Management process

6.3.5.1 Aspects of purpose

The Configuration Management (CM) process is used to establish and maintain needed control of the software system as it changes throughout its life cycle, including versions of its components and working code.

6.3.5.2 Aspects of outcomes and outputs

The outcome of CM is a software system whose components are known and located at specific points in time. Variances and discrepancies are known and managed. The principal focus of software CM is those items, products, or entities directly associated with the software for the project, i.e., the planning and engineering information in computer files, electronic media, and documents describing the software, and the computer files and electronic media containing the software itself. CM is also applied to the hardware and other elements of the SoI.

It is not the intention of ISO/IEC 12207:2017 to require all items, products, or entities to be managed with the same degree of rigor.

6.3.5.3 Related processes

CM can be applied to the work products and artifacts of any of the Technical processes. CM is closely allied with Information Management, which typically manages the content, structure, and format of information products.

CM, change management, and release management are often needed during a Transition process.

6.3.5.4 Activities, tasks, and approaches

Configuration strategy and configuration identification involves defining what to control and to what degree, considering risk, customer expectations, and legal and other constraints.

During the software life cycle, configuration control passes from the originator (author or developer) to the project or organization, and then (as agreed) to the acquirer. CM change control procedures identify the persons or groups with the authority to authorize and make changes at each level; and the steps to be followed to request authorization for changes, process change requests, track changes, distribute changes, and preserve past versions. CM is needed throughout the life cycle due to continuing updates and upgrades to the hardware and software environment and ongoing maintenance of the software SoI.

Rigorous change control procedures with multiple levels of review and approval slow the delivery cycle. Change control should be responsive when teams make many smaller changes that might not require the same level of scrutiny as a major release. To enable smaller, more frequent releases to production, an organization can establish a catalogue of the types of routine changes, whose risk has been pre-assessed. Stakeholders should be notified of configuration changes.

Software CM differs for software under development from more formal processes for baselined software. During development, the aim is to coordinate and control the work of multiple developers. Automated tools are used to merge code from a developer's branch into the working copy, and to prevent multiple developers from changing the same code at the same time. When developed code is tested and approved, a baseline is established for the software element.

The configuration identification scheme should include designators for entities to be placed under configuration control and should allow assigning a unique identifier to each software item. The designators should cover the software products to be developed or used and should cover the elements of the software engineering environment.

The configuration identification scheme should extend to the desired level of control needed for the technical process. Identification may include computer files, electronic media, documents, software

units, hardware items, and software items. The identification scheme should include the version, revision and release status of each configuration item. Configuration identification of the final software product may specify various elements of the baseline, including the source files and executable software (with batch files, command files, or other files needed to regenerate the executable software); a description of packaging requirements; compilation/build procedures for creating executable software; procedures for modifying the software; measured hardware resource utilization of the "as built" software; information for users, and a description of the software version.

Records should be prepared and maintained for entities that have been placed under a designated level of configuration control (e.g., project-level or higher configuration control). These records should be maintained for the duration of the agreement or according to organizational policy.

Two types of configuration audits can be performed – functional audit (FCA) and physical audit (PCA). These two audits have different purposes and methods.

A functional audit is used to demonstrate that system verification results compare favourably with the specifications against which verification was performed and that planned verification procedures were followed. This audit is also used to confirm that verification results compare favourably against configuration documentation such as drawings, authorized changes, and "as-built" or "as-coded" records.

A pre-production prototype or the first system produced is typically used for verification. This audit should be typically completed before release of the system for initial production.

A physical audit is performed to examine the "as-built" or "as-coded" system against its configuration documentation such as drawings, bill of materials, specifications, code lists, manuals, verification procedures, and acceptance data. The "as-built" or "as-coded" system examined should be one or more of the first set of systems produced during the initial production. Selection of the systems to be used in the audit should be done at random by the auditors. The purposes of the physical audit are as follows:

a) to confirm that the system has been realized correctly in accordance with its drawings or specifications;

b) to confirm that the information database represents the essential set of work products or artefacts from the engineering effort;

c) to confirm that required changes to previously completed specifications have been included;

d) to confirm that enabling systems for future system life cycle stages will be available, can be executed and meet stakeholder requirements;

e) to provide the basis for approval of further production of the system, if applicable.

6.3.5.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.3.5, ISO/IEC 19770 (all parts), IEEE 828, and ISO/IEC/IEEE 24748-8.

6.3.6 Information Management process

6.3.6.1 Aspects of purpose

Information management is used to acquire, transform, produce, maintain, and dispose of information needed by a project, organization, and stakeholders.

6.3.6.2 Aspects of outcomes and outputs

Information management determines the quality of technical information gathered or generated, the value of the developed information, and its timeliness, completeness, validity, confidentiality (if

required), and benefit to recipients. The information products (outputs) are generally associated with the processes that produce or modify them.

6.3.6.3 Related processes

All processes produce information to be managed. CM principles apply to information management, e.g., identifying items to be developed or controlled, identifying baselines, and obtaining the infrastructure and tools to manage information content.

The Disposal process applies to information as well as to software and information artifacts.

6.3.6.4 Activities, tasks, and approaches

The extent of information developed and retained should be based on project size, exit criteria for the life cycle stage, agreement deliverables, and applicability to other projects and products. Using the agreement and project plans and schedules, information managers are responsible for planning the following:

a) what information should be delivered to the acquirer and other stakeholders,

b) what information should be accessible to the acquirer and other stakeholders,

c) what information requires approval from the acquirer when first created or subsequently modified,

d) format and content requirements for the information, and

e) when information should be produced, delivered in preliminary and final form, and updated or revised.

NOTE Delivery of information can be considered as a related activity, communication management. It can include communications with all stakeholders, including the public, and escalation of communications to higher levels of the organization depending on the criticality of the message and the need for action.

Information items required or recommended by ISO/IEC/IEEE 12207:2017 are described in ISO/IEC/IEEE 15289:2019. Information should be tailored to the requirements of the organization, acquirer, and project to avoid developing, acquiring, and storing apparently useless data. The choice of whether to incorporate information directly or by reference to another document depends on several factors e.g., the ease of accessing the referenced document (especially during an emergency) and the provision of a single source content management system such that an update in one place is automatically applied to all the other documents where the same content is reused.

Deliverable data should have content, location, format, media, and packaging that is appropriate for the organization or project where it is developed, recorded, and used.

The retention period for records should be established, either directly or indirectly by reference to policies or legal requirements.

6.3.6.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.3.6, ISO/IEC/IEEE 26511, ISO/IEC/IEEE 26512, ISO/IEC/IEEE26513,ISO/IEC/IEEE26514,ISO/IEC/IEEE26515,ISO/IEC/IEEE23026,ISO/IEC/IEEE26531, ISO/IEC/IEEE 15289, and IEC/IEEE 82079-1.

6.3.7 Measurement process

6.3.7.1 Aspects of purpose

The software measurement framework should define a flexible measurement process that directly supports the software life cycle process and related tasks and activities as implemented for the project. The software measurement process should be used to effectively plan and track software activities and

tasks across the life cycle; aid in managing cost, schedule, quality, and technical risks; and integrate software measurement with existing program management and software programming disciplines established for the project.

6.3.7.2 Aspects of outcomes and outputs

Software product measures assess progress and achievement against system and other work product technical requirements. Software measurements are used for project estimation and performance improvement.

Results of validation are used to assess stakeholder satisfaction and to deliver improved value to the acquirers of project products and services.

Software measurement data should be made available, preferably by automated data access methods, to the acquirer and supplier.

This measurement framework should directly support the project software life cycle process and related tasks and activities. It should provide an overall structure for identifying and prioritizing software issues, selecting appropriate measures to address the identified issues, integrating the measurement requirements into the engineering and management processes, analyzing and reporting the measurement results, and taking appropriate action. The measurement framework should be implemented to support supplier and acquirer information throughout the project life cycle and should address the analysis of the feasibility of the project plans and the tracking of project performance against the plans. Measurement is an important consideration during the Operations process, to monitor performance trends and to evaluate the effect of changes in the operational schedule and procedures on performance versus cost.

A technical performance measure (TPM) is used to assess design progress, compliance to performance requirements, and technical risks for critical performance parameters. Selection of TPMs should be limited to critical technical thresholds or parameters that, if not met, put the project at cost, schedule, or performance risk. A TPM provides an early warning of the adequacy of a design in terms of satisfying selected critical technical parameter requirements. The TPM includes the projected performance, such as a growth curve with thresholds of acceptable variance.

6.3.7.3 Related processes

Measurement is part of the evaluation of every process, and particularly QA and Project Assessment and Control.

The Infrastructure Management process is applied to measurement systems and services.

6.3.7.4 Activities, tasks, and approaches

The measurement process includes the selection and analysis of appropriate software measures to address the specific project software issues, constraints, and objectives as determined by the acquirer and supplier. Measures should be limited to those few that will actually be used as the basis for decisions. The general recommendation is to use limited trials of a selected few measurements, then slowly expand once the reality of obtaining the information and actually using it for decisions is proven out. Measures should be re-evaluated for utility once they have been in place for a defined period.

6.3.7.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.3.8, ISO/IEC/IEEE 15939, and Function point standards such as ISO/IEC 14143.

6.3.8 Quality Assurance process

6.3.8.1 Aspects of purpose

Quality Assurance (QA) evaluates products and processes to help ensure compliance with predefined provisions and plans. QA assures that the agreed products specified in the contract exist, have undergone evaluations in accordance with the policies for conducting QA activities and tasks, and satisfy the acceptance criteria in the contract. QA tracks, treats, and resolves incidents and problems.

6.3.8.2 Aspects of outcomes and outputs

The outcome of QA is an independent assessment or evaluation using objective criteria that work was performed consistently with the organization's standards and procedures, or that the software system has been properly verified and validated to meet agreed levels of quality characteristics. Typical quality characteristics for software include functionality, reliability, usability, efficiency, maintainability, and portability.

6.3.8.3 Related processes

Quality Management sets the strategy and policies for the application of QA activities.

Common processes used to perform QA include Verification, Validation, and Measurement.

While QA controls incident and problem management, the actual work of solving problems occurs in many technical processes, especially Implementation, Operations, and Maintenance.

6.3.8.4 Activities, tasks, and approaches

For more reliable results, QA reviews and audits are customarily performed by someone other than the person who did the work. QA functions, if not provided by independent suppliers, should not be under the direct operational control of the project or product line manager and should report independently to a higher level of management or to QM.

QA should help coordinate related V&V processes so that unnecessary duplicate evaluations are avoided and that scheduled evaluations are mutually beneficial and realistic. While the quality of COTS or open source components may receive less intensive verification, the acquirer may still need assurance that these elements meet requirements.

NOTE Typical software reviews are peer reviews, walkthroughs, inspections, and joint reviews with stakeholders.

QA has the responsibility to coordinate and validate incident and problem resolution. Those responsible for the process or product are responsible for resolving problems found in those processes or products.

Incident management includes identification of the incident and determining if one or more incidents are related to a problem. Incidents and problems are categorized, investigated and analyzed if necessary, and prioritized for resolution. The status, planned and actual resolution of the incident or problem are communicated to stakeholders.

Incident management has different goals from problem management. The goal of incident management is to assist rapid recovery of services for the user, while retaining information needed for further investigation. The goal of problem management is to prevent the recurrence of related incidents by finding and eliminating the root cause(s) of the problem. When a problem analysis is extended, the problem can be tracked as a "known error" with available workarounds for the situation. Problem resolution and closure should include correction of associated procedures and information for users as well as corrections to the system.

6.3.8.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.3.8, ISO/IEC/IEEE 24748-8, IEEE 730, and ISO/IEC 90003. The ISO/IEC 25000 family of standards on Software product Quality Requirements and Evaluation (SQuaRE) provides detailed requirements and guidance.

6.4 Technical processes

6.4.1 Business or Mission Analysis process

6.4.1.1 Aspects of purpose

Business or Mission Analysis explores the area of opportunity and the feasibility for a project or product.

6.4.1.2 Aspects of outcomes and outputs

Business or Mission Analysis process can produce a context map and portfolio of opportunity areas with possible actions as an output. An operational concept and market analysis are usually outputs.

6.4.1.3 Related processes

The Infrastructure Management process is applied to enabling systems and services.

The Validation process is used to objectively confirm that the enabling system achieves its intended use for its enabling functions.

The CM process is used to establish and maintain configuration items and baselines.

6.4.1.4 Activities, tasks, and approaches

Iterative assessment includes decision criteria focused on the estimation of industrial feasibility of each alternative:

— elaborate possible system and operations concepts and system architectures and compare these against the identified needs, to determine levels of uncertainty and risks;

— assess the technical and programmatic feasibility of the possible concepts by identifying constraints relating to implementation, costs, schedules, organization, operations, maintenance, production, and disposal;

— identify critical subjects, technologies and propose pre-development activities;

— quantify and characterize critical elements for technical and economic feasibility;

— propose the system and operations concept(s) and technical solutions, including model philosophy and verification approach, to be further elaborated during next phase.

All disciplines affected by the system, software product, or software service to be acquired should be involved in concept definition. The operational concept should be modified/updated periodically to reflect current needs. The operational concept should address the full life cycle (acquisition, supply, implementation, operation, maintenance) of the system, software product, or software service. The description of the operational concept may be used for the following:

— obtain consensus among the acquirer, supplier, implementer, maintenance, and user agencies on the operational and maintenance concept for the proposed system;

— help users understand and adapt to operational and maintenance changes needed as a result of modifications to a currently operating system undergoing reengineering, addition of new capabilities, or other modifications; and

— form the basis of a risk assessment.

6.4.1.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.4.1 and ISO/IEC/IEEE 29148.

6.4.2 Stakeholder Needs and Requirements Definition process

6.4.2.1 Aspects of purpose

Stakeholder needs (including reports of deficiencies in existing systems) are obtained as inputs for functional requirements for new or modified software systems.

6.4.2.2 Aspects of outcomes and outputs

The primary outcome is a set of stakeholder requirements. A requirement is typically made up of what has to be done (a function) and how well it has to be done. A function is typically a statement with an actor (noun), an action (verb), and an object (noun) of the action. A non-functional requirement specifies how well the requirement is done, such as by software performance requirements, software external interface requirements, software usability, software design constraints, and software quality characteristics, which can be modeled to reflect data quality, product quality, and quality in use. Cost may be a requirement, either stated as a fixed cost (independent variable) or maximum cost (constraint). Informally, stakeholder requirements can be expressed in user stories.

EXAMPLE A stakeholder's functional requirement is for keyless door unlocking. A functional system requirement is "The software control module (actor) sends a signal to the electronic door lock (action), opening the door (object)". Non-functional requirements are for the system's software and hardware to use Wi-Fi and open the door within 250 ms.

6.4.2.3 Related processes

The Infrastructure Management process is applied to enabling systems and services for requirements management.

Functional and non-functional requirements may be identified in general terms by stakeholders and refined during Systems/software Requirements Definition.

The Validation process is used to objectively confirm that the enabling system achieves its intended use for its enabling functions.

6.4.2.4 Activities, tasks, and approaches

The acquirer and other interested parties together form the stakeholders related to the system being engineered. The acquirer stakeholder can be internal to the organization (for example another project, marketing organization, parent product team, the product team itself, user, operator, executive manager, supervisor), or external to the organization (for example a procurement agency, prime contractor, another organization, customer, user, operator, owner, purchaser). For a formal acquisition and agreement (contract), the acquirer provides the initial set of functional requirements for the software system.

It may not be possible to meet all stakeholder (acquirer and other interested party) requirements for a particular system, since various stakeholders have conflicting needs, requirements, and priorities. These conflicts should be identified and resolved during the performance of this process, or during the System/Software Requirements Definition process. Effectiveness assessments, trade-off analyses, and risk analyses should be used to resolve conflicts.

Information should be obtained from stakeholders about the context of use: information about the typical intended software users (including users of varying abilities) and the physical, technical, ethical, social, and cultural elements and working environment for a software system, which affect how the

software is used. It provides guidance to the software architects and designers on design alternatives. It is referenced in the design of validation activities (usability tests).

Sometimes other stakeholders can provide non-functional requirements for downstream system life cycle concerns, such as requirements for production, test, operations, logistics, deployment, training and information for users, maintenance, and disposal. Non-functional requirements are also derived through the System/Software Requirements Definition process. Non-functional requirements can include the following:

a) interfaces with associated enabling systems or interfaces with other systems in the intended operational environment;

b) critical factor needs, such as safety, security, reliability, availability, usability, and maintainability; as well as environmental and ethical considerations, local, national, and international regulations and laws.

c) operator and user needs, skills, competencies and working environments.

Measures of effectiveness (MOEs) should be explicitly identified for each software system and function. An MOE is an "operational" measure of success that is closely related to the achievement of the operational objective being evaluated, in the intended operational environment under a specified set of conditions; for example, how well the solution achieves the intended purpose. MOEs, which are stated from the user/customer viewpoint, are the customer's key indicators of achieving the objectives for performance, suitability, and affordability across the life cycle.

An initial set of stakeholder needs and requirements often contains contradictory, infeasible, or overlapping statements. Stakeholders should assist in prioritizing requirements and performance goals.

6.4.2.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.4.2; ISO/IEC/IEEE 29148, ISO/IEC 25010, ISO/IEC 25030, ISO/IEC 25063, and ISO 9241-11.

6.4.3 System/Software requirements definition process

6.4.3.1 Aspects of purpose

Since stakeholder requirements are not always stated in technical terms and readily usable for architectural design, the System/Software Requirements Definition process should be used to analyze the stakeholder requirements and transform them into a set of usable technical requirements. This includes the identification and analysis of external interface requirements, functional requirements, performance requirements, and constraints, as well as the quantitative and qualitative measures related to these requirements.

NOTE In this document, system requirements, software requirements, and system/software requirements cover the same scope, requirements for a software system. In other contexts, software requirements are regarded as the requirements met purely by software, without considering the hardware, communications, or infrastructure requirements.

System requirements should provide appropriate flexibility for designing, modifying, or expanding the delivered system. For example, in appropriate situations, system requirements may include an "open systems" approach. Alternatively, system requirements may call for use of, or consistency with, a domain specific architecture or the architecture of other software systems in the operating environment.

6.4.3.2 Aspects of outcomes and outputs

During System/Software Requirements Definition, the functional boundaries of the software system are established. Within those boundaries, the set of system/software requirements represents the accepted basis for the engineering of a software system. These requirements include the functions that

are required to be performed, how well they should be performed, the environment in which they are to be performed, and required characteristics of the system and services related to enabling systems.

Non-functional requirements specify how well the requirement is done, such as by software performance requirements, software external interface requirements, software usability, software design constraints, and software quality attributes, which can be modeled to reflect data quality, product quality, or quality in use.

6.4.3.3 Related processes

Requirements management is done in parallel with cost, schedule, quality, configuration, interface, risk and change management activities that track compliance with agreements and technical requirements.

The Infrastructure Management process is applied to enabling systems and services for requirements management.

Requirements are controlled using the CM process.

The Validation process is used to objectively confirm that the requirements are acceptable to the acquirer or user representative.

The Verification process is used to determine if requirements are well formed.

The System Analysis process can be used to assess feasibility, affordability, balance, and other requirements characteristics. The System Analysis process is also used to determine appropriate values for requirement parameters, considering the estimated cost, schedule, and technical performance of the software system.

6.4.3.4 Activities, tasks, and approaches

The level of detail in the system/software requirements is based on the level of knowledge about the operational concept and the interfacing systems and infrastructure. Requirements definition should be performed iteratively with the Architecture Definition and Design Definition processes. As the software system and its functions, sub-systems, and components, and required states or modes of operation are articulated, derived requirements become more evident. The software system requirements emerge and evolve as knowledge is gained by all parties involved.

Over time, changes in technology can lead to revised requirements to avoid product obsolescence or to the need to radically redevelop the system architecture.

Safety and security requirements should be considered for every software system, particularly those with a risk of hazard to persons, property, or the environment, and those storing PII. Requirements should be stated positively rather than attempting to prohibit all the things the software should not do. Safety and security requirements should detail protective or preventive features that prevent or reduce the risk of error and harm.

Each requirement should be stated in such a way that it can be objectively verified or validated. Requirements should be clear (unambiguous), correct, feasible to implement, and unique (not duplicated in the set). Each requirement statement should express only one requirement; otherwise it should be parsed into multiple simple requirements. Requirements sets should be analyzed for gaps, variance, and conflicts.

Traditionally, software requirements are stated in discrete sentences. Other methods of expressing functional requirements are prevalent, including user stories, scenarios, and use case diagrams that represent the user experience.

Software functional requirements are rarely detailed for every subsystem or component, but usually focus on a SoI, without detailing and retesting every requirement of the software operating system, database management system, web browser, and other open source and commercially available (COTS) software products and modules. Even the functional components are frequently met by open source or proprietary components whose features and requirements are known or considered acceptable. Also,

existing software and hardware usually has additional capabilities which are not explicitly needed for the SoI and are not stated as its requirements.

System/software requirements should include system performance, quantified in terms of resource utilization, number of concurrent users, response time, and availability. Constraints on performance requirements are due to the context of use, and can include maximum allowable use of processor capacity, memory capacity, input/output device capacity, primary and auxiliary storage device capacity, and communications/network speed.

Each measure of effectiveness (MOE) (see 6.4.2) has a corresponding set of measures of performance (MOPs). An MOP is a measure that characterizes physical or functional attributes relating to the software system operation. These technical performance indicators can be measured under specified testing or operational environment conditions. These attributes are considered as important to ensure that the system has the capability to achieve operational objectives. MOPs are used to assess whether the system meets design or performance requirements that are necessary to satisfy the MOEs from which they were derived. MOPs are derived from the solution provider's viewpoint and look at how well the delivered system performs against system level requirements, for example an aspect of the system performance or capability. MOPs often map to key performance requirements in the system specification. They are expressed in terms of distinctly quantifiable performance features, such as speed, range of acceptable input or output, or frequency. This requires careful definition of MOPs during the Software/System Requirements Definition process and during logical architecture design and also effective requirements tracing to ensure that the MOPs are in fact identified and designed into the system.

NOTE As detailed in the ISO/IEC 25000 family of standards, MOE for quality in use characteristics can be derived from stakeholder requirements and defined from user viewpoints. Additional measures for software system and product quality characteristics can be defined to help transform critical stakeholder requirements into system/software quality requirements.

Many methods are available for use in defining and managing system requirements (e.g., concept exploration and prototyping). Requirements management systems should allow for traceability of requirements from their source and to their realization in the architecture, design, and integrated software system.

6.4.3.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.4.3 and ISO/IEC/IEEE 29148. Models for requirements for software quality are provided in the ISO/IEC 25000 family of standards on Software product Quality Requirements and Evaluation (SQuaRE), including ISO/IEC 25012 and ISO/IEC 25030.

6.4.4 Architecture Definition process

6.4.4.1 Aspects of purpose

The System Architectural Design process is used to transform the defined set of technical requirements into an acceptable architectural solution that fulfils the technical requirements for the software system.

6.4.4.2 Aspects of outcomes and outputs

Architectural definition is concerned with developing satisfactory and feasible solutions for the set of derived system/software requirements, maintaining the integrity of the operational concept and reflecting the organizational objectives for the system. The architectural definition should be used throughout the system life cycle to predict and track fitness for use and for assessing changes to the system. The software system architecture represents the structure of the information system, the behavior of the various system elements and how they interoperate and interface with other systems. A consistent, coherent architecture helps avoid the inefficiency of multiple overlapping or incompatible solutions for different instances of the same functions. Good software architectures reflect quality characteristics through scalable, portable, resilient, secure, and maintainable solutions. Software

architectural definition should produce common mechanisms for managing persistent objects, providing a user interface, and managing storage.

The architectural definition should be documented in a technical data package or models or views that may include a set of architectural solution specifications and other configuration descriptions for the logical and physical architecture. Logical architecture artifacts can take one or more forms, such as the following:

a) a functional flow block diagram reflecting the decomposition of major functions into their sub-functions;

b) a data flow diagram that decomposes functions while explicitly showing the data needed for each function;

c) a data structure with corresponding functions and processing flows related to the data and associated with assigned technical requirements;

d) interface definitions with logical, physical, and functional attributes of system element and system to external system boundaries delineated;

e) a behavioural diagram that describes input stimuli and outputs by function and includes operating order, as appropriate to input or output criteria;

f) a control diagram that indicates the controlling factors of a function and the resulting behaviour;

g) the concept of execution: a description of the states and modes of the system;

h) a timeline that allocates a time requirement to a set of real-time functions;

i) a functional failure modes and effects table that indicates the possible effects of a function failure mode, such as not doing what it is designed to do or doing a function which it is not expected to perform, with possible resolutions for each failure mode;

j) objects that encapsulate a partition and mapping of technical requirements and that are characterized by services (behaviours, functions, and operations) provided by encapsulated attributes (values, characteristics, and data);

k) a set of algorithms derived from contextual diagrams.

NOTE IDEF0 (Integration DEFinition 0) function modelling is designed to represent the decisions, actions, and activities of an existing or prospective organization or system. See IEEE Std. 1320.1 for more information.

In a recursive approach, an outcome from the software architecture definition is the requirements for the more detailed software system design.

The logical architectural design solution selected may be defined to provide the physical architecture outputs:

a) a configuration description including the enabling systems needed to provide life cycle support to the designed system;

b) selection of appropriate platforms, information models, programming languages.

Configuration descriptions can be in the form of specifications, baselines, diagrams and other design descriptions. The specific configuration descriptions will depend on the life cycle stage in which the Architecture Definition process is being used, and the information needed to satisfy the exit criteria of the life cycle stage and the entry criteria for the next stage.

6.4.4.3 Related processes

Architecture definition is closely related to Design definition, as design constraints can affect the architecture alternatives.

Analysis of key drivers typically builds from the Business or Mission Analysis, Stakeholder Requirements Definition, and System/Software Requirements Definition processes. Definition of the software system context and boundaries is mainly based on the outcomes of the Business or Mission Analysis process and is performed concurrently with the Stakeholder Needs and Requirements Definition process.

The Infrastructure Management process supports use and reuse of enabling systems.

The Validation process is used to confirm that the architecture models and views reflect stakeholder requirements, that stakeholder concerns are addressed, and to help ensure that future iterations of software system architecture better address stakeholder concerns.

The CM process is used to establish and maintain architecture baselines.

The Information Management process controls the information items, such as architecture descriptions (architecture models, architecture views, evaluations, and traceability).

The Transition process is used to articulate interim and concurrent solutions while moving from a legacy architecture to a new one.

6.4.4.4 Activities, tasks, and approaches

ISO/IEC/IEEE 12207:2017 should not be interpreted as mandating a single architectural viewpoint and view. It does not require the use of any particular architectural framework, model, or method, e.g., functional decomposition or object-oriented design.

Acquirers and software development organizations can have different expectations for how and how much design detail should be documented and reviewed in the architecture definition, or top-level structure for a software system and each software element. To avoid misconceptions by various project stakeholders, the organizational policies and procedures or agreement with the acquirer should define the standards, methods, and tools used for architecture and design definition, including the following:

a) terminology and graphic notations to describe different design elements;

b) level of detail associated with architectural and detailed design;

c) approach for software V&V;

d) approach to reviewing design information with project participants.

Development of software architecture alternative solutions and selection of the preferred solution is based on analyses of the following characteristics:

a) feasibility of the physical interfaces (human, form, fit, function, data flow and interoperability with enabling systems and external systems);

b) degree to which attributes of security, safety, producibility, testability, ease of deployment, install ability, operability, supportability, maintainability, trainability, and disposability are capable of being designed in;

c) the variability and the sensitivity to variability for each identified critical performance parameter;

d) technology necessary to realize the solution effectively, the risks associated with introduction of new or advanced technologies, and alternative lower-risk technologies that could be substituted;

e) availability of off-the-shelf end products (such as reusable software) and the cost and risks in modifying an off-the-shelf system element to satisfy design and interface requirements;

f) effort to maintain a competitive system compared with potential or existing competitor products;

g) further design efforts that could be needed to build in secure features, accommodate redundancy, or support graceful degradation;

h) needs, requirements, and constraints for enabling systems;

i) capacity to evolve, or be re-engineered, incorporate new technologies, enhance performance, increase functionality or other cost-effective or competitive improvements, when the system is in production or in the marketplace;

j) limitations that can preclude the capability of the SoI or system element and related services to evolve (technology refresh or technology insert).

A software system architecture solution that involves humans and human constraints such as eye movement, information processing rates, and ergonomics should consider human usability factors.

Existing system elements, or the introduction of new technology, should be considered in establishing the logical architectural design models. The use of existing systems helps reduce developmental time and cost, but may increase complexity. The use of new technologies can provide a competitive edge, but can also increase risk. In such considerations, new interfaces may be introduced and should be included in the set of technical requirements through iteration of the System/Software Requirements Definition process.

During the architectural and design related processes, software architects should be aware that many requirements that call for development of a software product can probably be met by an existing open source or reusable software product. The reusable software product may be used as is or modified and may be used to satisfy part or the entire requirement. For example, a requirement may be met by using an existing plan, specification, or design.

The set of technical requirements can be allocated to logical architectural design models to form a set of derived technical requirements taking into consideration the operational environment. These derived technical requirements can be used as the basis for physical architectural design.

High-level software system architecture reflects system performance requirements (how well each function should be performed) without saying how that performance is to be achieved. If the result of the architectural analysis goes into more detail, including specific ways in which the performance should be achieved, this is termed a detailed design.

The development of the software architecture is aided by communications among the architects, software designers, and other key stakeholders to clarify requirements, their priorities, and the impact on the candidate solutions.

Modelling, simulation, and prototyping tools are typically used to evaluate the feasibility of fulfilling the requirements through the architecture and design. Tools and models also assist with selecting a software architecture that is more easily operated and maintained, e.g., as a model for a product line, expandable through a distributed configuration, or using encapsulation or containers for key functions.

Physical architecture design draws on the outputs of logical architecture definition and the two processes interact iteratively. In performing physical architecture design, the logical architectural design models, the derived technical requirements, and those technical requirements not allocated to logical architectural design models can be used to form alternative physical design solutions. After each alternative physical design is evaluated, the preferred architectural design solution should be selected using an appropriate analysis of cost effectiveness, operational effectiveness and risk.

During architecture definition, it can be necessary to re-engage the Stakeholder Needs and Requirements Definition and the System/Software Requirements Definition processes, if it is determined that requirements cannot be met because of unresolved issues or adverse cost, schedule, performance, or risk impacts.

6.4.4.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.4.4, ISO/IEC/IEEE 42010, ISO/IEC/IEEE 42020, and ISO/IEC/IEEE 42030.

6.4.5 Design Definition process

6.4.5.1 Aspects of purpose

Software design takes the structural, behavioral, and quality/performance elements of the requirements and software system architecture and shapes how they are implemented in more detailed models and software elements (features).

6.4.5.2 Aspects of outcomes and outputs

Detailed design results in the articulation of features, algorithms, patterns, and information models that will be represented in the working software. Detailed design considers the environment in which the SoI or system element should perform its functions, the interface and inter-changeability characteristics, and the methods for verifying or validating compliance with requirements. ISO/IEC/IEEE 12207:2017 does not require any specific notations or methods for software design.

It is not considered necessary to have a detailed design description of each software unit before beginning software implementation through coding or construction. When COTS software is acquired as part of the architecture, the detailed internal design of the COTS software may not be visible to the acquirer.

6.4.5.3 Related processes

Design definition is considered the extension of Architecture Definition.

For software, detailed design is often realized during software construction (Implementation). During detailed design, the System Analysis process should be applied to select the preferred methods, patterns, or algorithms for system realization.

The Verification process is performed at the same time so that a test method is available for each feature or component of the software design (test-driven development).

The Infrastructure Management process provides enabling systems and services.

The CM process is used to establish and maintain configuration items and baselines for artifacts such as design models.

The Information Management process controls the information items, such as design descriptions and specifications.

6.4.5.4 Activities, tasks, and approaches

The choice of how much detailed design to generate and document depends on the relative facility of the developers in carrying out a consistent architecture during the software construction or modification, customization, and integration. Detailed design also contributes to ease of subsequent refactoring or modification of the software. Detailed design specifications may be used to state design requirements in terms of how a requirement should be achieved and how a software element should be constructed and documented. Particular attention is needed for database access or manipulation, data elements and data element assemblies of the database. In another example, API usually have detailed design specifications to assist developers who are otherwise unfamiliar with the capabilities and internal functioning of a software system.

Rather than producing detailed software designs, agile methods often result in a 'just-enough' design, with the emphasis on rapid development of working software for selected functions. This practice is common when time to market is important or in maintenance fixes. However, minimizing design effort can lead to 'technical debt' affecting the rate of progress (velocity) in later iterations. Producing new features can reveal the need for significant rework of the existing code or database structure. Also, a design that is adequate for initial demonstration can be less resilient for high-volume operation or can be vulnerable to security risks.

Planning for detailed design should address the following:

a) standards, methods, and tools used;

b) terminology and graphic notations to describe different design elements;

c) level of detail associated with detailed design;

d) relationship of detailed design to separately compiled software entities and to separately executable software entities;

e) approach to reviewing design information with project participants; and

f) approach to unit testing.

Through successive iterations of the Architecture Definition and Design Definition processes, a software architecture and detailed design emerge. Further process application refines requirement allocation among software items until requirements are affiliated with realizable software units.

NOTE 1 ISO/IEC TR 19759 describes key issues in software design, including concurrency, control and handling of events, data persistence, distribution of components, error and exception handling and fault tolerance, interaction and presentation, and security. It also describes principles of user interface design.

The fully articulated design includes the selected means of implementation for each software element, such as new software, reused or refactored elements, open source or off-the shelf. The selection includes consideration of technical, cost, and schedule risks and trade-offs.

NOTE 2 Design-to-cost methods can be used to establish a maximum cost requirement equivalent to other system performance requirements.

Criteria that may be used in selecting software resources to realize a design include the following:

a) ability to provide required capabilities and meet required constraints;

b) ability to provide required safety, security, and privacy protection;

c) reliability and maturity, as evidenced by established track record;

d) testability;

e) interoperability with other system and system-external elements;

f) maintainability, including the following:

   1) likelihood the software product will need to be changed;

   2) feasibility of accomplishing that change;

   3) availability and quality of documentation and source files;

   4) likelihood that the current version will continue to be maintained by the producer;

   5) impact on the system if the current version is not maintained;

   6) warranties available.

g) short-term and long-term cost impacts of using the software product, including the following:

   1) the acquirer's usage and ownership rights to the software product;

   2) restrictions on copying and distributing the software or documentation; and

   3) license or other fees applicable to each copy.

6.4.5.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.4.5.

6.4.6 System Analysis process

6.4.6.1 Aspects of purpose

System analysis is used to increase understanding of some aspect of the software. Analysis is typically used to help make a decision or solve a problem by resolving it into smaller parts. Software analysis is used both in software design and in information design, either to diagnose problems in existing software and information, or to design new or refactored software elements.

6.4.6.2 Aspects of outcomes and outputs

The intended outcome is a result that solves the problem, or clarifies the assumptions, criteria, and components of the situation. Often the output of software analysis is a prototype, use case, algorithm, or pattern. Software analysis can involve factoring existing code into smaller, more easily understandable objects. Usually the expected outcome of software analysis is not just software or a system that performs a function, but which also achieves desired performance characteristics, such as reduced run time or resistance to hacking.

6.4.6.3 Related processes

Trade-off analyses are conducted throughout the Technical Management and Technical processes.

Systems analysis is typically used in conjunction with the Decision Management process.

The Infrastructure Management process enables the provision of systems analysis services.

The CM process is used to establish and maintain configuration items and baselines.

The Information Management process controls the information items. For this process, the analysis results or reports are typical information items that are managed.

6.4.6.4 Activities, tasks, and approaches

Trade-off analyses are conducted to resolve conflicts (such as conflicting requirements) and select a recommended solution from a set of defined alternatives (such as optional actions to take for risk resolution, resolutions for requirement conflicts, alternative logical architectural design solutions and alternative physical architectural design solutions). Outcomes from a trade-off analysis include the recommended option, implementation considerations, impacts related to each option, basis of recommendation and assumptions made.

The types of trade-off analyses typically performed during performance of life cycle processes include:

a) Formal: formally conducted, with results reviewed at technical reviews. Specific formal trade-off analyses are normally identified in an agreement.

b) Informal: follows the same methodology of a formal trade-off analysis but involves less documentation and review.

c) Judgmental: selection of a recommended option, based on judgement of the analyst or designer after a less rigorous analysis than that required by a formal trade-off analysis and for which the consequences are not as important. Used when one option is clearly superior to others or time is not available for a more formal approach. Most trade-off analyses performed are of the judgmental type.

Examples of candidate criteria that may be used in evaluating reusable software products include the following:

— ability to provide required capabilities and meet required constraints;

— ability to provide required safety, security, and privacy protections;

— testability;

— interoperability with other system and system-external elements;

— distribution issues, including the following:

  — restrictions on copying/distributing the software or documentation;

  — license or other fees applicable to each copy.

  — the acquirer's usage and ownership rights to the software product;

— maintainability, including the following:

  — reliability/maturity, as evidenced by established track record;

  — warranties available;

  — likelihood the software product will need to be changed;

  — feasibility of accomplishing that change;

  — availability and quality of documentation and source files;

  — likelihood that the current version will continue to be maintained by the producer;

  — impact on the system if the current version is not maintained;

— short-term and long-term cost impacts of using the software product;

— technical, cost, and schedule risks and trade-offs in using the software product.

ISO/IEC/IEEE 12207 does not venture into the many methods, patterns, and algorithms used for systems analysis used during software implementation (construction).

6.4.6.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.4.6.

6.4.7 Implementation process

6.4.7.1 Aspects of purpose

The Implementation process transforms software system requirements and design artifacts and patterns into elements of products or services.

6.4.7.2 Aspects of outcomes and outputs

An implemented system element can be either a single product or a composite product, depending on its position in the system structure and its source as modelled, built, acquired or reused.

NOTE The limits of the terms for software element, software component, feature, and software unit can vary in different organizations.

6.4.7.3 Related processes

Software implementation is usually performed concurrently with software integration. The strategy for software implementation involves System Analysis to select the approach.

The Verification process provides objective evidence that the software implementation fulfills its specified requirements and identifies anomalies (errors, defects, faults) in implemented artefacts.

The Validation process confirms that the implementation fulfils requirements for a specific intended use of a software work product.

CM supports any necessary configuration identification, control and change management during software unit construction and software element implementation.

Information Management controls required implementation-related information or documentation.

QA supports process and work product audits and inspections, and addresses problem, non-conformance, or incident reporting and handling.

The Project Planning and Project Assessment and Control processes to support planning and monitoring of reviews, audits, inspections, etc., to monitor critical processes and new technologies, and to analyse reported anomalies and their data, enable corrective actions and report lessons learned.

Software implementation commonly involves use of the Agreement processes to obtain non-developmental items (NDI).

6.4.7.4 Activities, tasks, and approaches

Specific activities and methods of software implementation depend on whether the product is new development (obtained by software construction), adaptation of off-the shelf or reused software, or adaptation of software services.

Software construction can be used to transform software element definitions into products or services. The implemented software element can be either a single product or a composite product, depending on its position in the system structure and its ability to be appropriately modelled, built, bought, or reused. The organization should set standards for the use of the CM system during software development and for consistent coding methods, such as naming of variables, error handling, and commenting. Often such standards can be enforced through the development tool suite.

The implementation strategy for construction identifies 'implement-to' criteria e.g. software architecture, requirements, design, test, documentation, CM, traceability, or other conditions to be satisfied. These criteria clarify appropriate unit aggregation levels, specifications, constraints, or other requirements and any special software life cycle development models, processes, approaches, methods, environments, etc. to be used for construction.

Database implementation almost always uses an existing database management application. Implementation may include configuring the database to suit the information model for the software, preparing data for use in development and testing, and populating the database or other data files with data values.

If the strategy is reuse, then the project will need to determine the level, source, and suitability of the reused system elements. The implementation strategy can include implementation processes and procedures, fabrication or construction processes, tools, methods and equipment, implementation tolerances or special handling, and verification uncertainties. In the case of repeated system element implementation (e.g. mass production, replacement system elements) the implementation procedures and fabrication processes are defined to achieve consistent and repeatable producibility.

Each software unit and affiliated data or other information (e.g. database or structures, test data, interface data, documentation, test procedures) is implemented according to the implementation strategy and criteria. Criteria for evaluation commonly include satisfaction of unit requirements and test criteria, usability for human interface functions, unit test coverage, traceability requirements,

consistency with software item requirements or design, internal unit requirement consistency, and feasibility for further process activity, e.g. integration, verification, validation, operations, and maintenance.

For systems that rely heavily on off-the-shelf system elements, implementation is often directed to start in the execution phase without doing concept studies. In this case, the project needs to be aware of the risks of starting implementation without doing the risk reduction engineering of earlier studies. Use of off-the-shelf system elements does not alleviate doing the engineering required to ensure system feasibility, compatibility of interfaces and interoperability of system elements. What this off-the-shelf approach does is to reduce the need to go through earlier decision gates, not eliminate the analysis necessary to reduce risks.

6.4.7.5 Related standard

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.4.7.

6.4.8 Integration process

6.4.8.1 Aspects of purpose

Software integration combines software and possibly additional hardware elements to produce a software system. Integration is most commonly used when the software system is a combination of COTS or reused elements and interfaces and new software. Integration also applies to embedded software (firmware) in hardware, such as an integrated processor module.

6.4.8.2 Aspects of outcomes and outputs

The outcome of integration is a software system ready for transition to operational use and interaction with other systems.

6.4.8.3 Related processes

Implementation and Integration are commonly executed as one process during software development, along with Verification. The Verification process provides objective evidence that the integrated software fulfils its specified requirements and to identify anomalies (errors, defects, faults) in integration-related information items, (e.g., system/software requirements, architecture, design, test, or other descriptions), processes, software elements, items, and units.

The Validation process confirms that a work product fulfils requirements for a specific intended use of an integrated software function.

Integration of humans and procedures into the system is typically performed during the Transition process. Integration occurs when software is migrated to a different hardware platform, but this is commonly treated as part of the Transition process.

The Quality Assurance process supports work product audits and inspections, and problem, non-conformance, or incident reporting and handling.

CM supports necessary configuration identification, control and change management.

Information Management supports required integration-related information or documentation.

The Project Assessment and Control process can be used in accordance with the integration strategy to plan and conduct technical reviews of the integrated software system elements, e.g., a test readiness review to help ensure the integrated element or system with its affiliated data and information items is ready for qualification testing.

The Infrastructure Management process is applied to enabling systems and services.

6.4.8.4 Activities, tasks, and approaches

Automated or continuous Implementation and Integration of software units and elements is supported by combined software development and test environments.

6.4.8.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.4.8 and ISO/IEC 24748-6.

6.4.9 Verification process

6.4.9.1 Aspects of purpose

Verification should be used to establish correspondence between the performance and characteristics of the software system and its technical requirements and other agreement requirements. This process helps ensure that each system element, system and the SoI of the system structure has been implemented or integrated correctly from the perspective of fulfilling its technical requirements.

6.4.9.2 Aspects of outcomes and outputs

The outcome of verification is a software system or element which has been shown to fulfil requirements.

6.4.9.3 Related processes

Verification can be applied at different points in the life cycle and is particularly applicable to requirements, architecture and design, integrated software, migration or transition, and maintenance.

CM supports configuration identification, control, and change management.

Information Management supports verification-related information or documentation.

The QA process supports work product audits and inspections, and handles problem, non-conformance, or incident reporting and management.

The Project Planning and Project Assessment and Control processes support planning and monitoring of verification activities (e.g. reviews, inspections, analyses, demonstrations, testing), schedules, available resources, and analysis of reported anomalies and their data, to enable corrective actions and report lessons learned.

6.4.9.4 Activities, tasks, and approaches

Verification should be planned. Verification can be dependent on the form of the system, on the decisions made regarding life cycle stage entry and exit criteria, as well as the specific life cycle stage. Planning for verification should include responsibilities for preparing, reviewing, approving, and updating plans, procedures, and reports. Verification needs an incident and problem management system to handle reported defects and errors. Verification planning should consider the likelihood that defects will be identified, requiring repeated software implementation, integration, and verification tasks (retests).

Example methods of verification include the following:

a) Inspection (for example, inspection of drawings).

b) Analysis (for example using mathematical modelling, simulation, a virtual reality prototype or similarity to a previously verified element). Simulation and modelling can be useful for studying system performance in the future operational environment and for saving costs when live load testing is impractical. An example of similarity is using already performed verifications of similar systems with similar configuration descriptions or systems that have already been certified to a standard.

c) Demonstration (for example using mock-ups, physical models or operation. An example of operation is the verification of configuration descriptions that apply to life cycle costs or system attributes such as MTBF).

d) Test (for example using physical products, prototypes).

For non-critical requirements, use of inspection, analysis (including simulations), and demonstration during any life cycle stage for verification can be useful to save cost. During early stages of a system life cycle, inspection, analysis, demonstration or similarity to previously verified items should be used for verification. For later stages, qualification testing before production or operational testing of the implemented or integrated system may be used for performing verification.

Preparation for testing should include the development of test plans, test acceptance criteria, test procedures (scripts) and test cases, as well as the test environment.

It is good practice to test the range of acceptable system inputs as well as testing error handling for out-of-range inputs.

For human-operated systems, usability testing including the information for users (documentation) should be performed to determine if the users can perform their tasks efficiently, effectively, and with satisfaction.

Software qualification is a summarization of the V&V activities, covering not only the software SoI itself, but also its interfaces with its environment. This activity helps ensure that each software element and the software system have been implemented or integrated correctly from the perspective of fulfilling its technical requirements. If qualification testing is a requirement for system acceptance, the supplier usually conducts a preliminary run of the qualification test cases and procedures to ensure the qualification test documentation is correct and the system performs as expected. The qualification may be concluded by an acceptance review or an operational readiness review.

Generally, verifications are conducted under controlled conditions to ensure that each configuration description requirement is satisfied by the system. Use of actual operational environments and use of operators for verification is uncommon. If the operational environment is a factor for a specific requirement, then it should be included in any modelling, simulation or other form of verification.

Verification should include each requirement. Since it is impractical to test software of any complexity end-to-end by executing every line of code, verification strategy depends on knowledge of the software system and its interrelated components. At certain gateways it may be decided to perform comprehensive regression testing of all functions, the most commonly used functions, or especially problematic functions. During maintenance, judgment is needed to determine whether verification is needed only for the modified component, or whether regression tests should be conducted.

Failure of software qualification testing can result from poor conduct of the verification or improper implementation or integration of the system. Anomalies that are discovered during verification of the system (or system element or SoI) need to be appropriately resolved prior to the transition of the system to the acquirer and before performing software acceptance. Re-verification tests should be repeated when test outcome variations and anomalies are traced to poor verification conduct or to inadequate verification environment.

Reports of verification activities should document the results, including the completion status of each scenario or test case; root cause of anomalies if known; and an overall assessment of the software, identifying remaining deficiencies, limitations, constraints and associated impacts as well as recommendations for correction.

6.4.9.5 Related standards

This subclause relates to ISO/IEC/IEEE 12207:2017, 6.4.9; IEEE 1012, ISO/IEC/IEEE 29119 (all parts), ISO/IEC/IEEE 26513, and ISO/IEC 25062.