INTERNATIONAL STANDARD

ISO/IEC 26553

First edition
2018-11

Information technology — Software and systems engineering — Tools and methods for product line realization

Technologies de l'information — Ingénierie des systèmes et du logiciel — Outils et méthodes pour la réalisation d'une gamme de produits

Reference number
ISO/IEC 26553:2018(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.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).

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 of the voluntary nature of standards, 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 Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*.

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 main purpose of this document is to deal with the capabilities of methods and tools of software and systems product line (SSPL) realization which includes detailed design and implementation. This document defines how the tools and methods can support the software and systems product line-specific realization processes.

Domain realization will be carried out based on domain architecture that provides structures and constraints that govern the subsequent SSPL lifecycle processes. The outcomes of domain realization processes are transferred into the realization of a member product at the application realization processes. Therefore realization support tools and methods should consider both engineering processes, namely domain realization and application realization.

Product line realization can be differentiated from a single product development because of the following aspects:

— The outcomes of domain requirements engineering and domain architecture form the basis for product line realization unlike the case of a single product development. There are two core processes in product line realization: domain realization and application realization. The major aims of the domain realization processes are to conduct detailed design and further implementation based on domain architecture, which includes commonality and variability for a family of products, and to prepare necessary variability information for variability modelling. Whereas, the major aims of the application realization processes are to conduct detailed design and implementation for application realization and to bind variability whose defined binding time is realization stage.

This document can be used in the following modes:

— by the users of this document: to benefit people who conduct detailed design and implementation for software and systems product lines;

— by a product line organization: to provide guidance on the evaluation and selection for methods and tools for product line realization; and

— by providers of methods and tools: to provide guidance on implementing or developing tools and methods by providing a comprehensive set of the capabilities of tools and methods for product line realization.

The ISO/IEC 26550 family of standards addresses both engineering and management processes and capabilities of methods and tools in terms of the key characteristics of product line development. This document provides processes and capabilities of methods and tools for product line realization. Other standards in the ISO/IEC 26550 family are as follows:

ISO/IEC 26550, ISO/IEC 26551, ISO/IEC 26555, ISO/IEC 26557, ISO/IEC 26558 and ISO/IEC 26559 are published. ISO/IEC 26552, ISO/IEC 26554, ISO/IEC 26556, ISO/IEC 26560, ISO/IEC 26561, ISO/IEC 26562 and ISO/IEC 26563 are planned International Standards. The following list provides an overview of the series:

— processes and capabilities of methods and tools for domain requirements engineering and application requirements engineering are provided in ISO/IEC 26551;

— processes and capabilities of methods and tools for domain design and application design are provided in ISO/IEC 26552;

— processes and capabilities of methods and tools for domain testing and application testing are provided in ISO/IEC 26554;

— processes and capabilities of methods and tools for technical management are provided in ISO/IEC 26555;

— processes and capabilities of methods and tools for organizational management are provided in ISO/IEC 26556;

— processes and capabilities of methods and tools for variability mechanisms are provided in ISO/IEC 26557;

— processes and capabilities of methods and tools for variability modeling are provided in ISO/IEC 26558;

— processes and capabilities of methods and tools for variability traceability are provided in ISO/IEC 26559;

— processes and capabilities of methods and tools for product management are provided in ISO/IEC 26560;

— processes and capabilities of methods and tools for technical probe are provided in ISO/IEC 26561;

— processes and capabilities of methods and tools for transition management are provided in ISO/IEC 26562;

— processes and capabilities of methods and tools for configuration management of asset are provided in ISO/IEC 26563; and

— others (ISO/IEC 26564 to ISO/IEC 26599) are to be developed.

Information technology — Software and systems engineering — Tools and methods for product line realization

1 Scope

This document, within the context of tools and methods of detailed design and implementation for software and system product lines:

— provides the terms and definitions specific to realization for software and systems product lines;

— defines processes performed during product line realization (those processes are described in terms of purpose, inputs, tasks and outcomes);

— defines method capabilities to support the defined tasks of each process; and

— defines tool capabilities to automate/semi-automate tasks or defined method capabilities.

This document concerns processes and capabilities of realization tools and methods for a family of products, not for a single system.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

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

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

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

3.1
application component
component that is selected, reused or newly developed for a *member product* (3.14)

3.2
application configuration
structure of a *member product* (3.14), including *application components* (3.1) and *application interfaces* (3.3)

3.3
application interface
interface that is selected, reused or newly developed by a *member product* (3.14)

3.4
application realization
one of the application engineering processes that includes detailed design and implementation

3.5
application-specific component
component that is developed for a specific *member product* (3.14)

3.6
aspect
special consideration within product line engineering process groups and tasks with which specialized methods and tools can be associated

3.7
binding time of variability
stage when the value of variability is determined

3.8
component implementation
activity of realizing a component, including unit test

3.9
configuration parameter
parameter provided by variable components or interfaces, so that its value is selected when bindings occur

3.10
domain component
reusable component among *member products* (3.14) within a product line

3.11
domain interface
reusable interface among the components of a *member product* (3.14) within a product line

3.12
domain realization
one of the domain engineering processes that include detailed design and implementation

3.13
extractive approach
approach of developing the initial baseline of a product line from one or more existing products

3.14
member product
product belonging to the product line

3.15
proactive approach
approach of developing an innovative product line or product variations based on organizational predictions that anticipate a stated product need

3.16
reactive approach
approach of developing a product line or product variations in response to stated needs or customer requirements

3.17
texture
architectural texture
collection of common development rules and constraints for realising the applications of a product line

3.18
variability implementation
variability development in source codes or executable modules

4 Reference model for product line realization

4.1 Overview

Product line realization supports the detailed design and implementation of a product line. This document provides requirements and guidance for building reusable components and member product-specific components. Product line realization consists of two separated processes: domain realization and application realization. Domain realization develops reusable components and interfaces that will be reused in application realization. COTS (commercial off-the-shelf), open sources and/or licensed third-party platforms can also be a significant part of domain assets together with the organizations' own domain artefacts. Many components are derived from domain components by selecting variants while member product-specific components or interfaces are built in application realization. However, not all planned components and interfaces are built in domain realization. Hence proper strategies for domain and application realization are necessary to enhance the reusability and to raise cost efficiency of the product line. Organization realization management provides subprocesses that deal with these at the organizational level. Annex A describes the scope of realization activities.

Realization of a product line shall be conducted using the common rules produced during the architecture design. Architectural textures provide rules and constraints that realization should adhere to, so that domain and application realization are coordinated well with each other.

NOTE 1 The architectural texture is the collection of common rules for realizing the system, such as coding rules and general mechanisms. Styles and design patterns are examples of the texture. Architectural texture demands the use of a hierarchy of layers, subsystems and components. It can require the use of the façade pattern or the presence of an interface with a prescribed set of functions as each component.

The reference model specifies the structure of supporting processes and subprocesses for product line realization. As shown in Figure 1, product line realization can be structured into seven processes; *organizational realization management, detailed domain design, domain implementation, variability management in realization, asset management in realization, detailed application design and application implementation*. Each process is divided into subprocesses to address product line realization issues, and each subprocess is described in terms of the following attributes:

— the title of the subprocess;

— the purpose of the subprocess;

— the outcomes of the subprocess;

— the inputs to produce the outcomes;

— the tasks to achieve the outcomes; and

— the capabilities of methods and tools required for performing the tasks effectively and efficiently.

NOTE 2 When the process, subprocess, outcomes and tasks are listed or described in a sentence they are italicized in order to increase their visibility.

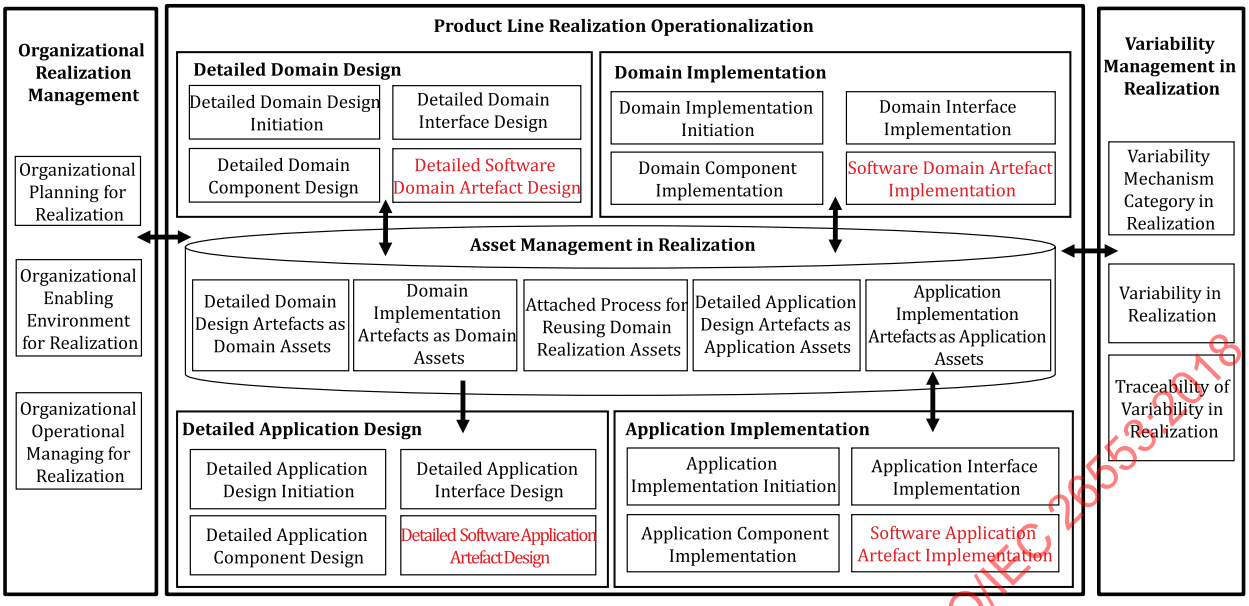

Figure 1 — Product line realization reference model

An organization's life-cycle models and strategy guide its realization process in a proactive approach, a reactive approach or an extractive approach. According to the predictability of variations among member products, those approaches can be applied, i.e. when the variations are predictable, a proactive approach is suitable, whereas when they are not, a reactive approach or an extractive approach can be used. Though the variations are not predictable, the reuse of the existing assets (or COTS components), should be considered during components or interfaces realization.

4.2 Organization realization management

The organizational realization management provides realization-specific managerial supports such as plans, enabling environments and operations. Organizational realization management shall serve to do the following and to define the capabilities of tools and methods for supporting them:

— *Organizational planning for realization* establishes an organization level plan to proceed to product line realization from product line architecture design based on targeted product line requirements.

— *Organizational enabling environment for realization* establishes environments required for domain realization (detailed domain design and implementation), for deriving the realization of a specific member product from domain realization, and for the additional realization of product-specific requirements. The enabling environment includes organizational structure, technologies and tools that support product line realization.

— *Organizational operational managing for realization* monitors and controls the status of product line realization for harmonizing works conducted in domain engineering and application engineering processes.

4.3 Domain realization

The domain realization encompasses the detailed design and implementation of domain components and interfaces. Detailed domain design shall serve to do the following and to define the capabilities of tools and methods for related support:

— *Detailed domain design initiation* starts the detailed domain design by following domain architecture design specification.

— *Detailed domain interface design* elaborates on the detailed design of the interfaces of common and variable components to expose required and/or provided functionality of the components.

— *Detailed domain component design* elaborates the details of components such as classes and objects to be decomposed into data structures and algorithms based on the reference architecture and texture.

— *Detailed software domain artefacts design* conducts detailed design for software domain artefacts' functionalities that will be commonly used by member products of a product line.

4.4 Domain implementation

The domain implementation builds components and interfaces to be commonly used or to be selectively used by member products. Domain implementation includes building and buying components and supporting infrastructure. The planned variability should be realized with adequate variability mechanisms, and core components and interfaces are validated based on the domain architecture. Domain implementation shall serve to do the following and to define the capabilities of tools and methods for related support:

— *Domain implementation initiation* starts the domain implementation by following detailed domain design results.

— *Domain interface implementation* realizes domain interfaces based on the detailed interface design.

— *Domain component implementation* realizes domain components (e.g., coding, compiling, linking and loading in the case of software systems) based on the detailed design of the component.

— *Software domain artefacts implementation* produces software files for software domain artefacts' functionalities that will be commonly used by member products of a product line.

The difference between domain realization and a single system realization is in that the reusable components of domain realization are loosely coupled, configurable, and may not be executable because of variability.

4.5 Asset management in realization

Major domain assets in realization are detailed design artefacts, implementation artefacts and executable components. Application assets produced from application realization should be managed for application. In addition, application assets may be a part of core assets after being reviewed and reengineered by domain engineers. The domain asset management in realization shall serve to do the following and to define the capabilities of tools and methods for related support.

— *Detailed domain design artefacts as domain assets* make detailed domain design models and specifications conform to the domain asset structure that includes attributes to describe domain assets and index/annotation to retrieve or trace them.

— *Domain implementation artefacts as domain assets* make domain components and domain interfaces, which are declared in programming language files, conformant with the domain asset structure.

— *Attached process for reusing domain realization assets* defines processes that should be adhered to when a member product reuses domain realization assets.

— *Detailed application design artefacts as application assets* establish and maintain detailed application design models and specifications for each member product to be referred to for later maintenance and evolution.

— *Application implementation artefacts as application assets* establish and maintain member product-specific components and interfaces including executable artefacts for each member product to be referred to for later maintenance and evolution.

4.6 Detailed application application design

Application realization encompasses the detailed design and implementation of application components and interfaces. The detailed application design shall serve to do the following and to define the capabilities of tools and methods for related support:

— *Detailed application design initiation* initiates detailed application design that follows application architecture.

— *Detailed application interface design* elaborates on the detailed design of the interfaces of application components to expose required and/or provided functionality of the components.

— *Detailed application component design* elaborates the details of application components such as classes and objects to be decomposed into data structures and algorithms.

— *Detailed software application artefacts design* conducts detailed design for application-specific software application artefacts' functionalities.

4.7 Application implementation

The application implementation builds member product-specific components and interfaces. The application implementation shall serve to do the following and to define the capabilities of tools and methods for related support:

— *Application implementation initiation* srarts application implementation by following detailed application design. In order to start application implementation, implementation configurations are derived from domain implementations and detailed application design should be ready for guiding application implementation.

— *Application interface implementation* implements application interfaces in accordance with the detailed design specification.

— *Application component implementation* implements application components in accordance with the detailed design specification.

— *Software application artefacts implementation* produces software files for application-specific software application artefacts' functionalities.

4.8 Variability management in realization

Variability in domain design is distributed over domain components and interfaces in detailed design. Variability can appear in manifold manners in domain assets, so they make the variability management complicated, particularly if they show widespread impacts. Variability mechanisms in detailed design and variability mechanisms in implementation are quite different, so there are two different variability managements in realization, i.e. variability management in detailed design and variability management in implementation. Variability can be handled in a late stage of the member product lifecycle (e.g. building time or run-time) as this offers high flexibility. The variability management in realization shall serve to do the following and to define the capabilities of tools and methods for related support:

— *Variability mechanism category in realization* identifies and classifies variability realization mechanisms used in realization stage.

— *Variability in realization* deals with the capabilities to locate and implement variants in interfaces and variants over components. This subprocess deals with variability model and variability mechanisms in domain realization.

— *Traceability of variability in realization* establishes and maintains trace links between variability and realization artefacts (e.g. detailed design artefacts, codes and object files) including trace links between different abstraction levels of variability.

The identification and analysis of the key differentiators between single-system engineering and management and product line engineering and management can help organizations to understand the product line and to formulate a strategy for successful implementation of product line engineering and management. The key aspects have been defined in ISO/IEC 26550 and Table 1 shows the category of the key aspects.

Table 1 — Key aspects for identifying product line-specific realization tasks

Category	Aspects
Reuse management	application engineering, domain assets, domain engineering, product management, platform, reusability
Variability management	binding, variability
Complexity management	collaboration, configuration, enabling technology support, reference architecture, texture, traceability
Quality management	measurement and tracking, cross-functional verification and validation

For product line realization, relevant processes and tasks shall be identified on the basis of these aspects. The concerns specific to product line realization will enable an organization to understand the product line realization relevant processes, subprocesses, tasks, methods and tools' capabilities. The following describes each aspect concerning product line realization.

— **Application engineering**. Application realization is the task of application engineering where domain components and interfaces are reused and member product-specific components and interfaces are implemented.

— **Binding**. Binding time of variability is important for the flexibility of a product line. Variability can be introduced in realization and their binding times should be determined by considering trade-offs. During domain realization, variability bindings may occur at the defined binding times, and decisions for the binding mechanisms of internal variability may be made.

— **Collaboration**. Because it is very important that domain architecture and architectural texture are adhered to and large numbers of designers and developers are involved in domain realization, proper methods for coordinating and aligning designers and developers should be devised.

— **Configuration**. As bindings occur and member product-specific components and interfaces are added during realization, multiple versions of components and interfaces are generated. Their configurations should be managed considering the entire product line.

— **Domain asset**. Domain components and interfaces are the major assets of domain realization. Domain components and interfaces, including variability models, should be properly managed and an asset base should be used to support their reuse.

— **Domain engineering**. Domain realization is the task in domain engineering where domain components and interfaces are designed and coded and, if possible, compiled and built.

— **Enabling technology support**. Technologies for managing the complexity of detailed domain designs, coding and building should be supported.

— **Measurement and tracking**. Domain components and interfaces should be measured and monitored from the viewpoint of the overall product line objectives for on time corrective actions.

— **Platform**. Components and interfaces that constitute the platform of a product line are developed during domain realization.

— **Product management**. Since member products within a product line evolve continuously in accordance with the changes of markets and business opportunities, product management should monitor and support the evolution of domain realization.

— **Reference architecture**. The reference architecture is the major input for domain realization. The reference architecture provides the list of components and interfaces that should be developed during domain realization.

— **Reusability**. For enhancing reusability of domain components, variability should be properly distributed to components and interfaces.

— **Texture**. Texture produced during domain design provides common guidelines to realization by specifying common rules and constraints for correct domain/application component and interface realization.

— **Traceability**. Traceability between domain/application design artefacts and domain/application realization artefacts, between domain/application realization artefacts and domain/application variability model in realization, and between domain variability model in realization and application variability model in realization, should be established and maintained.

— **Cross-functional validation and verification**. Provisioning of objective evidences for validation and verification of domain realization, application realization and variability model in realization is an important aspect.

— **Variability**. Variability is realized through components, interfaces, a set of classes, single class or lines of code.

5 Organizational realization management

5.1 General

Domain realization performs detailed domain design and implementation from domain architecture and architectural texture. Architectural texture, which is defined through domain architecture processes, should be adhered to by domain and application realization. The organizational realization management offers integrated realization environments and managerial supports so that domain/application designers and developers perform their works under the well-coordinated and managed ways. An organizational realization management process includes three key subprocesses:

— *Organizational planning for realization* establishes and maintains domain and application realization plans to guide and control the overall product line realization endeavours.

— *Organizational enabling environment for realization* establishes and maintains interoperable environments with those of the previous stage and among varying environments that support realization activities.

— *Organizational operation management for realization* monitors and controls the overall progress of product line realization.

5.2 Organizational planning for realization

5.2.1 Principal constituents

5.2.1.1 Purpose

The purpose of this subprocess is to establish a realization capability required for realizing components and interfaces from the outcomes of the previous lifecycle stages. Domain architecture, application architecture and their process artefacts are reviewed, and proper plans to precede realization should be established based on the review results. Plans for realization should be harmonized with the plans established in the organizational management of ISO/IEC 26556.

5.2.1.2 Inputs

The following inputs should be available to perform the organizational planning for realization process:

— organizational transition plans;

— domain assets configured in domain architecture stage;

— application assets configured in application architecture stage; and

— quality assurance results for architecture design stage.

5.2.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the organizational planning for realization process:

— *Organizational plans for detailed domain design* are defined.

— *Organizational plans for domain implementation* are defined.

— *Organizational plans for detailed application design* are defined.

— *Organizational plans for application implementation* are defined.

5.2.1.4 Tasks

The organization shall implement the following tasks with respect to the organizational planning for realization process:

— *Confirm the readiness of realization*: help ensure that domain/application architecture processes have been completed as planned and the resulting artefacts satisfy the required quality.

— *Define realization plans*: establish plans describing the realization subprocess, purposes, outcomes, activities, tasks, resources and actors and to making preparation for tracking its progress.

5.2.2 Confirm the readiness of realization

The goal of this task is to assure that the previous processes have been conducted enough to proceed to the domain/application realization process.

The method should support confirming the readiness of realization with the following capabilities:

— making use of entry criteria which have to be assured to proceed to domain/application realization;

— making use of entry criteria of domain realization which have to be assured to proceed to application realization (e.g. example entry criterion can be defined as “rules and constraints adhered to in application realization for the right reuse of domain realization artefacts should be defined”);

— making use of checklists to evaluate the readiness; and

— reviewing selected artefacts to confirm the readiness.

A tool should support confirming the readiness of realization by allowing the user to do the following:

— share defined entry criteria;

— report confirmed entry criteria; and

— access the artefacts to be evaluated.

5.2.3 Define realization plans

The goal of this task is to define realization plans including work plans, risk mitigation plans and operation plans. A work plan includes a plan for acquiring and adapting COTS components.

The method should support defining plans for realization with the following capabilities:

— defining plans for domain/application realization in accordance with the readiness evaluation results; and

— making preparation for traking the actual progress based on plans.

A tool should support defining plans for realization by allowing the user to do the following:

— access the templates of organizational-level product line plans;

— share organizational-level product line plans defined in the organizational management of ISO/IEC 26556;

— share plans with the relevant stakeholders; and

— track progress against the plans.

5.3 Organizational enabling environment for realization

5.3.1 Principal constituents

5.3.1.1 Purpose

The purpose of this subprocess is to provide integrated and interoperable enabling environments for within and between domain and application realization processes. Enabling environments include hardware, software, skills, procedures, methods, techniques and tools necessary to execute domain and application realization. Particularly, this process deals with interoperability among tools that support domain and application realization, including tools for variability management.

5.3.1.2 Inputs

The following inputs should be available to establish the organizational enabling environment for the realization process:

— procedures and estimated resources necessary to execute realization; and

— organizational plans for realization.

5.3.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the organizational enabling environment for the realization process:

— *Enabling environment requirements in realization* are identified and defined.

— *Enabling environment for realization* are maintained and improved.

5.3.1.4 Tasks

The organization shall implement the following tasks with respect to the organizational enabling environment for the realization process:

— *Analyse requirements for enabling environments*: identify and define infrastructure necessary to proceed to domain and application realization.

— *Establish and maintain enabling environments*: maintain and improve the defined infrastructure required for executing domain and application realization.

— *Enable interoperability among related infrastructure elements*: allow interoperation with those hardware, software, procedures, methods and tools for domain and application architecture, domain realization and application realization and variability modeling.

5.3.2 Analyse requirements for enabling environments

The goal of this task is to collect needs for infrastructure required for carrying out realization plans.

The method should support analyzing requirements for enabling environments with the following capabilities:

— reviewing gathered requirements for infrastructure;

— resolving conflicts among requirements for infrastructure;

— documenting rationale for conflict resolution; and

— documenting requirements for infrastructure.

A tool should support analyzing requirements for enabling environments by allowing the user to do the following:

— utilize browsing, reviewing and commenting environments; and

— use templates for documenting requirements for infrastructure.

5.3.3 Establish and maintain enabling environments

The goal of this task is to establish the required infrastructure and maintain and improve the infrastructure.

The method should support establishing and maintaining enabling environments with the following capabilities:

— supporting acquisition of infrastructural elements (e.g. detailed design support tool, IDE, unit testing tool);

— providing integration of infrastructural elements; and

— allowing maintenance of stable infrastructure.

A tool should support establishing and maintaining enabling environments by allowing the user to do the following:

— integrate infrastructure elements; and

— use a communication channel for suggesting improvement on infrastructure.

5.3.4 Enable interoperability among related infrastructure elements

The goal of this task is to enable interoperable environments that support domain/application realization delivering realization artefacts from domain/application architecture. The interoperability requirements among tools used in adjacent product line lifecycle processes are defined and addressed.

Enabling environments support application realization delivering application realization artefacts from domain realization artefacts (e.g. domain design models, components, interfaces). The interoperability includes environments required for binding variants in application.

The method should support enabling interoperability among related infrastructure elements with the following capabilities:

— allowing consistent domain realization with domain architecture and architectural texture;

— allowing consistent application realization with application architecture and architectural texture;

— linking variability management in domain realization with variability management in domain architecture;

— allowing integrated domain realization and application realization through variability binding;

— ensuring availability of binding information (including rules and constraints) at right place and time for application realization;

— allowing consistent and integrated variability model reference in design and realization;

— providing variability model view in realization; and

— providing seamless link between domain variability models in realization and application realization.

A tool should support enabling interoperability among related infrastructure elements by allowing the user to do the following:

— exchange data (e.g. models) for proceeding to domain realization from domain architecture design (this deals with data exchange among domain processes, application processes and variability models);

— exchange data for proceeding to application realization from application architecture design;

— support seamless proceeding of variability management from domain architecture design to domain realization;

— use binding information in order to correctly derive application realization from domain realization;

— perform variability binding for the consistent derivation of concrete realization in the application realization process;

— establish seamless link with variability model in design for revising, decomposing or adding variability;

— access variability model at realization stage; and

— select relevant domain/application realization artefacts.

5.4 Organizational operational managing for realization

5.4.1 Principal constituents

5.4.1.1 Purpose

The purpose of this subprocess is to monitor and control product line realization progress. Measures are defined for monitoring and controlling the realization progress.

5.4.1.2 Inputs

The following inputs should be available to perform the organizational operational managing for the realization process:

— detailed domain/application design status information (e.g. planned versus actual, resource consumed); and

— domain/application implementations status information.

5.4.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the organizational operational managing for the realization process:

— *Measures* are defined for monitoring and controlling the realization progress.

— *Data for measures* are collected.

— *Corrective action list including its status for realization* is established.

— *Capability improvement* is realized.

5.4.1.4 Tasks

The organization shall implement the following tasks with respect to the organizational operational managing for realization process:

— *Monitor and control progress in realization*: measure the actual realization progress coordinated with the defined business values.

— *Make corrective action and improvement in realization*: perform product line evolution, deployment and innovation in realization.

5.4.2 Monitor and control progress in realization

The goal of this task is to check the progress of domain and application realization based on operational plans against their assigned roles and responsibilities.

The method should support monitoring and controlling progress in realization with the following capabilities:

— defining measures for monitoring and controlling domain/application realization progress based on the assigned business values;

— defining monitoring and controlling points including relations among those points located in domain realization or application realization; and

— measuring and integrating the realization progress conducted in different organization units.

A tool should support monitoring and controlling progress in realization by allowing the user to do the following:

— collect and integrate data (semi-)automatically;

— generate adequate progress reports (of plans); and

— share monitoring and controlling information with relevant realization participants.

5.4.3 Make corrective action and improvement in realization

The goal of this task is to activate corrective actions in realization, so that a product line organization improves its capability for achieving its business value.

The method should support making corrective action in realization with the following capabilities:

— decomposing corrective actions into roles and responsibilities for domain and application transferred from the higher level of management; and

— assigning roles and responsibilities to the relevant participants of realization.

A tool should support making corrective action in realization by allowing the user to do the following:

— perform simulation for confirming the right assignment of roles and responsibilities; and

— let relevant participants know their roles and responsibilities for delivering corrective actions.

6 Detailed domain design

6.1 General

In this process, detailed attributes and methods for components and interfaces are declared. Detailed domain design shall be consistent with the domain architecture and the architectural texture. Internal variability can be added or previously defined variability can be refined during detailed domain design. Detailed domain design process includes the following key subprocesses:

— *Detailed domain design initiation* sets up the readiness for proceeding detailed domain design process.

— *Detailed domain interface design* executes low-level designs for interfaces.

— *Detailed domain component design* executes low-level designs for components.

— *Detailed software domain artefact design* develops and documents a detailed design for databases, protocols, data streaming formats and threads.

6.2 Detailed domain design initiation

6.2.1 Principal constituents

6.2.1.1 Purpose

The purpose of this subprocess is to initiate detailed domain design. Initiation tasks include decisions related to the selected COTS and the qualification of detailed domain design capabilities such as tools, using detailed design approaches, and required internal technologies.

6.2.1.2 Inputs

The following inputs should be available to perform the detailed domain design initiation process:

— COTS selected in domain architecture;

— domain requirements specification;

— domain architectural structure; and

— domain architectural texture.

6.2.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the detailed domain design initiation process:

— *COTS to be tailored* are selected.

— *Detailed domain design capabilities* are established.

— *Detailed domain design* is initiated.

6.2.1.4 Tasks

The organization shall implement the following tasks with respect to the detailed domain design initiation process:

— *Review COTS from detailed design perspective*: revisit and evaluate selected COTS components to check whether they will be reused as it is or not.

— *Confirm inputs for detailed domain design*: revisit and evaluate inputs required for proceeding detailed domain design.

— *Confirm detailed domain design capability*: check whether required detailed design capabilities are established.

6.2.2 Review COTS from detailed design perspective

The goal of this task is to make proper decision for selecting or modifying appropriate components with low costs.

The method should support reviewing COTS from detailed design perspective with the following capabilities:

— defining evaluation perspective (approach) for COTS selection in detailed design;

— supporting cost estimation necessary for tailoring or modifying COTS; and

— supporting evaluation for fitness of COTS under consideration.

A tool should support reviewing COTS from detailed design perspective by allowing the user to do the following:

— access information for the selected COTS component during domain architecture design;

— perform data collection and calculation for cost estimation; and

— describe rationale related to decision.

6.2.3 Confirm inputs for detailed domain design

The goal of this task is to help ensure the quality of inputs required for the further proceedings of detailed domain design task.

The method should support confirming inputs for detailed domain design with the following capability:

— defining quality criteria from detailed domain design perspectives.

A tool should support confirming inputs for detailed domain design by allowing the user to do the following:

— access information for inputs; and

— record rationale related to decisions.

6.2.4 Confirm detailed domain design capability

The goal of this task is to check whether detailed domain design can be initiated properly.

The method should support confirming detailed domain design capability with the following capability:

— providing confirmation aspects for ensuring the readiness (e.g., plans, procedures, standards, textures, tools, resources) of detailed domain design.

A tool should support confirming detailed domain design capability by allowing the user to do the following:

— access information for the established organizational level enabling environments.

6.3 Detailed domain interface design

6.3.1 Principal constituents

6.3.1.1 Purpose

The purpose of this subprocess is to declare common and variable interfaces for components. Many interfaces can be provided for a component or they can be related with multiple components.

6.3.1.2 Inputs

The following inputs should be available to perform the detailed domain interface design process:

— domain requirements specification; and

— reference architecture specification.

6.3.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the detailed domain interface design process:

— *Evaluation results of detailed domain interface design* are documented.

— *Detailed design specification of domain interfaces* are established.

6.3.1.4 Tasks

The organization shall implement the following tasks with respect to the detailed domain interface design process:

— *Examine interactions among domain components*: analyse the configuration parameters and the return values of interactions among the relevant domain components for finding services to be exposed.

— *Define the detailed internal structures of domain interfaces*: design methods provided or required by domain interfaces.

— *Verify and validate detailed domain interface design*: help ensure that the noncompliance issues against product line architecture and errors in detailed domain interface design are identified and resolved.

— *Document detailed domain interface design*: specify the detailed domain interface design.

6.3.2 Examine interactions among domain components

The goal of this task is to capture the details of interactions between components and find domain services to be exposed.

The method should support examining interactions among domain components with the following capabilities:

— revealing interactions between domain components;

— investigating common and variable services to be exposed; and

— examining variability included in interactions.

A tool should support examining interactions among domain components by allowing the user to do the following:

— access domain architecture design;

— visualize relationships among domain components; and

— visualize variability included in interactions.

6.3.3 Define the detailed internal structures of domain interfaces

The goal of this task is to define common and variable services including their signatures that will be used as component interfaces.

The method should support the definition of the detailed internal structures of domain interfaces with the following capabilities:

— differentiating variable interfaces from common interfaces;

— providing variability mechanisms used at interface design;

— providing a guidance that supports variability distribution to interfaces;

— reviewing binding time issues that should be decided at interface design; and

— defining signatures for common or variable interfaces.

A tool should support the definition of the detailed internal structures of domain interfaces by allowing the user to do the following:

— utilize explicit notation in detailed design for differentiating variable interfaces from common interfaces; and

— use detailed design environment for domain interfaces.

6.3.4 Verify and validate detailed domain interface design

The goal of this task is to confirm whether the detailed domain interface design adheres to the rules defined in domain architecture specification and help ensure the correctness of the detailed domain interface design.

The method should support verifying and validating the detailed design of domain interfaces with the following capabilities:

— designing test requirements, test cases and test procedures (these will be reused for integration testing after refinement) for variability-related interface verification; and

— supporting validation for conformances among variability-related interfaces.

A tool should support verifying and validating the detailed design of domain interfaces by allowing the user to do the following:

— access domain architecture specification;

— describe variability in designing test requirements, test cases and test procedure; and

— send feedback for the identified non-conformance issues to related domain engineering process.

6.3.5 Document detailed domain interface design

The goal of this task is to specify and document detailed domain interface design.

The method should support documenting detailed domain interface design with the following capabilities:

— providing domain interface specification language;

— defining a documentation template that covers variability in detailed design of domain interfaces; and

— providing placeholders in a template for variability.

A tool should support documenting detailed domain interface design by allowing the user to do the following:

— utilize the defined domain interface description language;

— use a template for documentation; and

— import the detailed design of domain interfaces, including variability in the form of being easy to be substituted.

6.4 Detailed domain component design

6.4.1 Principal constituents

6.4.1.1 Purpose

The purpose of this subprocess is to elaborate components so that they can be reused correctly in different member products including details of declared provided and required interfaces. Different components can be designed for each variant of variability or a component can address variability.

6.4.1.2 Inputs

The following inputs should be available to perform the detailed domain component design process:

— domain requirements specification;

— domain architecture specification; and

— detailed domain component and interfaces design specification.

6.4.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the detailed domain component design process:

— *Evaluation results of detailed domain component design* are documented.

— *Detailed design specification of domain components* is established.

6.4.1.4 Tasks

The organization shall implement the following tasks with respect to the detailed domain component design process:

— *Define detailed internal structures of domain components*: design internal structures for using relevant variable interface, internal variability and variable component.

— *Verify and validate detailed domain component design*: help ensure that noncompliance issues against domain architecture, errors in detailed domain component designs and non-conformance with related domain interfaces are identified and resolved.

— *Prepare test inputs for unit testing*: generate test inputs such as test cases and test data with test engineers.

— *Document detailed domain component design*: specify the detailed design of domain components.

6.4.2 Define the detailed internal structures of domain components

The goal of this task is to design domain components in accordance with the common and variable services defined in domain interfaces.

The method should support defining the detailed internal structures of domain components with the following capabilities:

— providing variability mechanisms used at component design;

— providing a guidance that supports variability distribution to components;

— reviewing binding time issues that should be decided at component design;

— expressing explicitly components that include variability or variable components; and

— visualizing commonality and variability in domain component diagrams.

A tool should support defining the detailed internal structures of domain components by allowing the user to do the following:

— utilize explicit notation for expressing components that include variability; and

— use detailed design environment for domain components.

6.4.3 Verify and validate detailed domain component design

The goal of this task is to confirm whether the detailed design of domain components adheres to the rules defined in domain architectural texture and help ensure the correctness of detailed domain component design.

The method should support verifying and validating detailed domain component design with the following capabilities:

— designing test requirements, test cases and test procedures (these will be reused for integration testing after refinement) for variability related component verification against relevant interfaces;

— supporting validation for conformances among variability related components;

— validating the testability of detailed domain component design related to variability; and

— evaluating quality attributes considering related variability in detailed design level.

A tool should support evaluating detailed domain component design by allowing the user to do the following:

— access domain requirements specification;

— access domain architecture specification;

— describe variability in designing test requirements, test cases and test procedures; and

— send feedback for the identified non-conformance issues to related domain engineering process.

6.4.4 Prepare test inputs for unit testing

The goal of this task is to design test cases and test data that will be used in unit testing. Test engineers that will test domain components should participate.

The method should support preparing test inputs for unit testing with the following capabilities:

— manipulating variability in test requirements, test cases and test procedures for unit testing; and

— assuring testability for components.

A tool should support preparing test inputs for unit testing by allowing the user to do the following:

— access domain architecture texture; and

— describe variability included in test requirements, test cases and test procedures.

6.4.5 Document detailed domain component design

The goal of this task is to make detailed design specification of domain components.

The method should support documenting detailed domain component design with the following capabilities:

— defining a documentation template that covers variability in detailed domain component design; and

— providing placeholders in a template for variability.

A tool should support documenting the detailed domain component design by allowing the user to do the following:

— utilize a template for documentation; and

— import detailed domain component design with variability in the form of being easy to be substituted.

6.5 Detailed software domain artefact design

6.5.1 Principal constituents

6.5.1.1 Purpose

The purpose of this subprocess is to make detailed software domain artefact design besides domain interfaces and domain components. Software domain artefact include database, data communication, protocol and threads.

6.5.1.2 Inputs

The following inputs should be available to perform the detailed software domain artefact design process:

— domain requirements specification; and

— domain architecture specification.

6.5.1.3 Outcomes

The following outcomes shall be available as the results of the successful implementation for the detailed software domain artefact design in the domain process:

— *Evaluation results for detailed software domain artefacts* are documented.

— *Detailed design specification for software domain artefacts* is established.

6.5.1.4 Tasks

The organization shall implement the following tasks with respect to the detailed software domain artefact design process:

— *Define Detailed software domain artefact design*: elaborate detailed design to other artefacts.

— *Verify and validate Detailed software domain artefact design*: help ensure that noncompliance issues against domain architecture and errors in detailed domain software artifact designs are identified and resolved.

— *Prepare test inputs for unit testing*: generate test inputs such as test cases and test data with test engineers.

— *Document Detailed software domain artefact design*: specify the detailed design of domain software artifact.

6.5.2 Define Detailed software domain artefact design

The goal of this task is to design software domain artefacts in accordance with the basic services (i.e. database, communications, threads) defined in domain architecture.

The method should support defining detailed software domain artefact design with the following capabilities:

— providing variability mechanisms used at detailed software domain artefact design;

— providing a guidance that supports variability distribution to detailed software domain artefacts;

— explicitly expressing detailed software domain artefacts that include variability or variable artefacts; and

— visualizing commonality and variability in each artefact (e.g. visualizing variability in data models for database).

A tool should support defining detailed software domain artefact design by allowing the user to do the following:

— utilize explicit notation for expressing variability for each type of software domain artefacts; and

— use detailed design environment for producing detailed design artefacts for each common software artefact's functionality.

6.5.3 Verify and validate detailed software domain artefact design

The goal of this task is to confirm whether the detailed design of software domain artefacts adheres to the rules and constraints defined in domain architecture and texture, and help ensure the correctness of detailed software domain artefact design.

The method should support verifying and validating detailed software domain artefact design with the following capabilities:

— supporting validation for conformances with domain architecture;

— validating the testability of detailed software domain artefact design related to variability; and

— evaluating quality attributes considering related variability in detailed software domain artefact design.

A tool should support verifying and validating detailed software domain artefact design by allowing the user to do the following:

— access domain architecture specification and texture; and

— send feedback for the identified non-conformance issues to related domain engineering process.

6.5.4 Prepare test inputs for unit testing

The goal of this task is to design test cases and test data that will be used in unit testing. Test engineers that will test domain software artifacts should participate.

The method should support preparing test inputs for unit testing with the following capabilities:

— designing test requirements, test cases and test procedures (these will be reused for integration testing after refinement) for variability related to software domain artefact verification; and

— supporting validation for conformances among variability related to software domain artefacts.

A tool should support preparing test inputs for unit testing by allowing the user to do the following:

— access domain architecture texture; and

— allow the description of variability in designing test requirements, test cases and test procedure.

6.5.5 Document detailed software domain artefact design

The goal of this task is to make a specification for detailed software domain artefact design.

The method should support documenting detailed software domain artefact design with the following capabilities:

— defining a documentation template that covers variability in detailed domain software artifacts design; and

— providing placeholders in a template for variability.

A tool should support documenting the detailed software domain artefact design by allowing the user to do the following:

— utilize a template for documentation; and

— import detailed software domain artefact design of with variability in the form of being easy to be substituted.

7 Domain implementation

7.1 General

Domain implementation makes tangible and executable (after binding in the case of variable one) common and variable components including relevant interfaces (in case of software domain, implementation means writing codes). Domain implementation can include variability, so all components including interfaces are not concrete. During domain implementation, components that consist of software, system and subsystems are loosely coupled with each other. Domain implementation process includes the following key subprocesses:

— *Domain implementation initiation* sets up the readiness for proceeding domain implementation process.

— *Domain interface implementation* produces tangible and executable common and variable interfaces.

— *Domain component implementation* produces tangible and executable (after binding in the case of variable one) common and variable components.

— *Software domain artefact implementation* produces software files for the common software domain artefact functionalities.

7.2 Detailed domain implementation initiation

7.2.1 Principal constituents

7.2.1.1 Purpose

The purpose of this subprocess is to initiate domain implementation. Initiation tasks include decision related to qualifying domain implementation capabilities such as tools, use of domain implementation approaches and necessary technologies.

7.2.1.2 Inputs

The following inputs should be available to perform the detailed domain design initiation process:

— detailed domain design specification; and

— required domain implementation capabilities.

7.2.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the detailed domain design initiation process:

— *Domain implementation capabilities* are established.

— *Domain implementation* is initiated.

7.2.1.4 Tasks

The organization shall implement the following tasks with respect to the detailed domain design initiation process:

— *Confirm inputs for domain implementation*: revisit and evaluate inputs required for proceeding domain implementation.

— *Confirm domain implementation capability*: check whether required domain implementation capabilities are established.

7.2.2 Confirm inputs for domain implementation

The goal of this task is to help ensure the quality of inputs required for proceeding domain implementation.

The method should support confirming inputs for domain implementation with the following capability:

— defining quality criteria from domain implementation perspectives.

A tool should support confirming inputs for domain implementation by allowing the user to do the following:

— access information for inputs; and

— record rationale for decision.

7.2.3 Confirm domain implementation capability

The goal of this task is to check whether domain implementation capability is adequate to initiate domain implementation immediately .

The method should support confirming domain implementation capability with the following capability:

— providing confirmation aspects for ensuring the readiness (e.g., plans, procedures, standards, textures, tools, resources) of domain implementation.

A tool should support confirming domain implementation capability by allowing the user to do the following:

— access information for the established organizational level enabling environments.

7.3 Domain interface implementation

7.3.1 Principal constituents

7.3.1.1 Purpose

The purpose of this subprocess is to implement all interfaces that are declared in the interface design. In accordance with the defined strategy, the level of interface detail should be compromised.

7.3.1.2 Inputs

The following inputs should be available to perform the domain interface implementation process:

— detailed design specification for domain interfaces.

7.3.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the domain interface implementation process:

— *Written source codes for domain interfaces* are produced.

— *Executable files of domain interfaces* are produced.

7.3.1.4 Tasks

The organization shall implement the following tasks with respect to the domain interface implementation process:

— *Implement domain interface*: make common interfaces concrete based on the detailed domain interface design.

— *Build domain interfaces*: make domain interfaces executable.

— *Verify and validate domain interface implementation*: help ensure that implemented interfaces adhere to the constraints defined in architectural texture and detailed design specifications and errors in interface implementation are removed.

7.3.2 Implement domain interface

The goal of this task is to write codes for domain interfaces based on the detailed design of domain interfaces.

The method should support the implementation of domain interface with the following capabilities:

— providing variability mechanisms (related to interfaces) used at implementation;

— reviewing binding time issues that should be decided at implementation; and

— expressing commonality and variability at code level.

A tool should support implementing domain interface by allowing the user to do the following:

— utilize a coding or parts implementation environment that supports the expression of common and variable interfaces.

7.3.3 Build domain interfaces

The goal of this task is to create executable domain interfaces. Building task and verifying/validation task should be conducted iteratively.

The method should support building domain interfaces with the following capabilities:

— providing variability mechanisms (related to interfaces) used at build time; and

— building interfaces that include variability.

A tool should support building domain interfaces by allowing the user to do the following:

— use an environment that builds common and variable interfaces.

7.3.4 Verify and validate domain interface implementation

The goal of this task is to confirm whether domain interfaces are implemented consistently and correctly against the detailed domain design and whether domain interfaces adhere to the constraints/rules defined at domain architecture design.

The method should support verifying and validating domain interface implementation with the following capabilities:

— executing verification for domain interfaces related to variability; and

— validating interface conformances related to variability.

A tool should support verifying and validating domain interface implementation by allowing the user to do the following:

— verify for domain interfaces related to variability;

— record verification and validation results;

— track the status of identified non-compliance issues by separating commonality and variability; and

— send feedback for the identified non-conformance issues to related domain engineering process.

7.4 Domain component implementation

7.4.1 Principal constituents

7.4.1.1 Purpose

The purpose of this subprocess is to implement components that provide the commonality and required variability. Domain component implementation should be conducted in accordance with domain implementation strategy.

7.4.1.2 Inputs

The following input should be available to perform the domain component implementation process:

— detailed design specification of domain components.

7.4.1.3 Outcomes

The following outcome shall be available as a result of the successful implementation of the domain component implementation process:

— *domain components* (e.g. source code and executable in case of software) are produced.

7.4.1.4 Tasks

The organization shall implement the following tasks with respect to the domain component implementation process:

— *Implement domain components*: make components (write codes in the case if software) based on the detailed domain components design.

— *Build domain components*: make domain components tangible and executable.

— *Verify and validate domain component implementation*: help ensure that implemented components adhere to the constraints/rules defined in architectural texture and detailed design specifications and errors in domain components are removed.

— *Integrate domain components*: integrate domain components and COTS based on domain architecture.

7.4.2 Implement domain components

The goal of this task is to make concrete domain components based on the detailed design of domain components.

The method should support the implementation of domain components with the following capabilities:

— providing variability mechanisms (related to components) that can be specialized at implementation;

— reviewing binding time issues that should be decided at component implementation;

— explicitly expressing variability included in component implementation; and

— allowing easy configuration in member products.

A tool should support the implementation of domain components by allowing the user to do the following:

— refer variability mechanisms in component implementation; and

— establish links between variants and related components to allow easy configuration.

7.4.3 Build domain components

The goal of this task is to create executable domain components. In the case of software product line this task includes compile, linking and loading. Building task and verifying/validation task should be conducted iteratively.

The method should support building domain components with the following capabilities:

— providing variability mechanisms (related to components) that can be specialized in build time; and

— expressing explicitly variability in build time.

A tool should support building domain components by allowing the user to do the following:

— utilize an environment that builds common and variable components.

7.4.4 Verify and validate domain component implementation

The goal of this task is to confirm whether domain components are implemented consistently and correctly against the detailed domain design and whether domain components adhere to the constraints/rules defined at domain architecture design. Verification includes unit testing for a component.

The method should support verifying and validating domain component implementation with the following capabilities:

— providing ways to handle variability included in a component for verification (e.g. unit testing);

— executing verification for domain components related to variability; and

— validating conformances against related domain interfaces.

A tool should support verifying and validating domain component implementation by allowing the user to do the following:

— verify for domain components related to variability;

— record verification and validation results;

— track the status of identified non-compliance issues by separating commonality and variability; and

— send feedback for the identified non-conformance issues to related domain engineering process.

7.4.5 Integrate domain components

The goal of this task is to make the possible integration of platform based on the defined domain architecture.

The method should support integrating domain components with the following capabilities:

— allowing loosely coupled integration for variable interfaces; and

— allowing loosely coupled integration for variable components.

A tool should support integrating domain components by allowing the user to do the following:

— utilize integration environment for domain; and

— interact with integration testing tools.

7.5 Software domain artefact implementation

7.5.1 Principal constituents

7.5.1.1 Purpose

The purpose of this subprocess is to implement software domain artefact that provides the commonality and required variability. Software domain artefact implementation should be conducted in accordance with domain implementation strategy.

7.5.1.2 Inputs

The following input should be available to perform the software domain artefact implementation process:

— detailed design specification of software domain artefacts.

7.5.1.3 Outcomes

The following outcome shall be available as a result of the successful implementation of the software domain artefact implementation process:

— *Software domain artefacts* (e.g. database, data communication, threads) are produced.

7.5.1.4 Tasks

The organization shall implement the following tasks with respect to the software domain artefact implementation process:

— *Implement software domain artifacts*: materialize software domain artefacts (write codes in the case if software) based on the detailed software domain artefact design.

— *Build software domain artefacts*: make software domain artefacts ready to execute.

— *Verify and validate software domain artefact implementation*: help ensure that implemented software domain artefacts adhere to the constraints/rules defined in architectural texture and detailed design specifications and errors in software domain artefacts are removed.

7.5.2 Implement software domain artefacts

The goal of this task is to make concrete software domain artefacts based on their detailed design.

The method should support the implementation of software domain artefacts with the following capabilities:

— providing variability mechanisms (related to software domain artefacts) that can be specialized at implementation;

— expressing explicitly variability included in software domain artefact implementation; and

— allowing easy configuration in member products.

A tool should support the implementation of software domain artefacts by allowing the user to do the following:

— refer variability mechanisms in software domain artefact implementation; and

— establish links between variants and related software domain artefacts to allow easy configuration.

7.5.3 Build software domain artefacts

The goal of this task is to create executable software domain artefacts. In the case of software product line this task includes compile, linking and loading. Building task and verifying/validation task should be conducted iteratively.

The method should support building software domain artefacts with the following capabilities:

— providing variability mechanisms (related to software domain artefacts) that can be specialized in build time; and

— expressing explicitly variability in build time.

A tool should support building software domain artefacts by allowing the user to do the following:

— utilize an environment that builds common and variable software domain artefacts.

7.5.4 Verify and validate software domain artefacts

The goal of this task is to confirm whether software domain artefacts are implemented consistently and correctly against the detailed design and whether they adhere to the constraints/rules defined at domain architecture design. Verification includes unit testing for a software artefacts.

The method should support verifying and validating software domain artefacts with the following capabilities:

— providing ways to handle variability included in a software domain artefact for verification (e.g. unit testing);

— executing verification for software domain artefacts related to variability; and

— validating conformances against relavant software domain artefacts.

A tool should support verifying and validating software domain artefacts by allowing the user to do the following:

— verify software domain artefacts related to variability;

— record verification and validation results;

— track the status of identified non-compliance issues by separating commonality and variability; and

— send feedback for the identified non-conformance issues to related domain engineering process.

8 Variability management in realization

8.1 General

The majority of variability is materialized through variability mechanisms that can be used in the realization stage. Moreover, internal variability can be newly introduced or predefined variability can be refined. Thus variability models should be refined and trace links should be refined accordingly. In addition variability mechanisms used to introduce variability should be properly documented in order to provide necessary information to application realization. Variability management in realization process includes the following key subprocesses:

— *Variability mechanism category in realization* maintains variability mechanism category suitable for variability bound at realization stage;

— *Variability in realization* describes variability in realization explicitly and document variability in realization; and

— *Traceability of variability in realization* establishes and maintains trace links between variability and realization artefacts.

8.2 Variability mechanism category in realization

8.2.1 Principal constituents

8.2.1.1 Purpose

The purpose of this subprocess is to establish and maintain variability mechanism category specific to realization.

Decisions for many of the variabilities and their binding times are made at design time, and SSPL realization conducts detailed design and implementation in accordance with the decision, i.e. architectural structure and texture. Variability in the realization stage is realized by variability-enabling mechanisms such as inheritance, overloading/overriding, configuration parameters, polymorphism, parametric polymorphism and so on. Particularly, a lot of configuration parameters are defined at the realization stage for implementing variability and decision sets for the post-implementation time, i.e. compile time and run time are also defined. Such variability mechanisms can be classified into three categories; SSPL specifically defined mechanism, Language extension mechanism and Language support mechanism. The following list provides the exemplar variability mechanism category in realization:

— *SPL specifically defined mechanism:* variability realization mechanisms in Common Variability Language description (existence, substitution, value assignment and opaque).

— *Language extension mechanism:* the use of UML (unified modelling language) extensions (stereotype, tagged value and notes).

— *Language support mechanism:* inheritance, polymorphism, parameterized class, generics in codes, pointcut and advice concepts in AOP (aspect-oriented programming), framework implementation methods (e.g. hotspots and hooks) in framework.

8.2.1.2 Inputs

The following inputs should be available to perform the variability mechanism category in the realization process:

— guidance for variability mechanism selection (ISO/IEC 26557); and

— variability mechanisms in realization (elicited from from variability mechanism pool).

8.2.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the variability mechanism category in the realization process:

— *Variability mechanisms used for implementing variability in realization* are categorized.

— *Guidance is tailored specific to realization stage*.

— *Usage status of variability mechanism category in realization* is traced.

8.2.1.4 Tasks

The organization shall implement the following tasks with respect to the variability mechanism category in the realization process:

— *Identify variability mechanisms in realization by category*: find variability mechanisms suitable for variability in realization.

— *Guide the use of variability mechanism category in realization*: support correct use of variability mechanism.

— *Trace the usage status of variability mechanism category in realization*: check the efficiency and effectiveness of variability mechanisms.

— *Update variability mechanism category in realization*: improve variability mechanism category.

8.2.2 Identify variability mechanisms in realization by category

The goal of this task is to find variability mechanisms suitable for the characteristics of domain and variability in realization. Variability mechanisms should be reviewed depending on the decisions for architectural structure, texture and enabling environment.

The method should support identifying variability mechanisms in realization by category with the following capabilities:

— reviewing the available variability mechanisms to determine whether they comply with the explored architectural decisions and textures in realization time;

— assuring whether a variability mechanism fully covers the defined variability including dependencies and constraints;

— assuring whether a variability mechanism fully covers the defined variability including dependencies and constraints that should be bound in the implementation stage; and

— evaluating ease of binding and configuration.

A tool should support identifying variability mechanisms in realization by category by allowing the user to do the following:

— refer detailed design level variability mechanisms pool with their usage guidance;

— refer detailed implementation level variability mechanisms pool with their usage guidance for selection; and

— store rationales for variability mechanism selection.

8.2.3 Guide the use of variability mechanism category in realization

The goal of this task is to provide guidance for selection and decisions for binding time and configuration mechanisms, and performing binding in a member product. The overall guidance is provided in ISO/IEC 26557. This task supports concrete guide creation filled with realization-specific practices.

The method should support guiding the use of variability mechanism category in realization with the following capabilities:

— tailoring variability mechanism selection criteria specific to realization stage;

— tailoring detailed procedures for variability mechanism selection and use in realization stage;

— tailoring configuration guides and rules in realization stage.

A tool should support guiding the use of variability mechanism category in realization by allowing the user to do the following:

— share guides and rules for variability mechanism operation with relevant participants; and

— access guides and rules instantly for performing variability mechanism operation.

8.2.4 Trace the usage status of variability mechanism category in realization

The goal of this task is to trace the effectiveness and efficiency of variability mechanisms identified in category.

Measures and metrics used for tracing the usage status of variability mechanisms are defined, and the usage status are measured and the results should be properly analysed for providing valuable feedback.

The method should support tracing the usage status of variability mechanism category in realization with the following capabilities:

— confirming whether variability mechanisms in realization (e.g. variability mechanisms in codes) can correctly realize the defined variability dependencies;

— confirming whether variability mechanisms in realization can correctly realize the defined constraints;

— providing evaluation algorithm for verifying whether a variability mechanism satisfies the defined variability dependencies and constraints; and

— verifying the testability of variability mechanisms used in realization.

A tool should support tracing the usage status of variability mechanism category in realization by allowing the user to do the following:

— visualize variability mechanisms in realization;

— verify whether the variability mechanism properly supports the defined variability including dependencies and constraints; and

— verify testability of variability mechanisms used in realization.

8.2.5 Update variability mechanism category in realization

The goal of this task is to improve the established variability mechanisms by category.

Variability mechanisms in realization identified by category can be re-organized and improved based on analysis results of their usage status.

The method should support updating variability mechanism category in realization with the following capabilities:

— preparing items for variability mechanism category improvement;

— improving variability mechanism category; and

— propagating improvement results to variability mechanism pool.

A tool should support updating variability mechanism category in realization by allowing the user to do the following:

— share improvement items for variability mechanism category among appropriate participants;

— share improved variability mechanism category; and

— perform automatic propagation of improvement results to variability mechanism pool.

8.3 Variability in realization

8.3.1 Principal constituents

8.3.1.1 Purpose

The purpose of this subprocess is to address variability with the interfaces, components and source codes.

The architect usually only determines what interfaces exist, what their role is and which components should provide or require the interface. The most important constraint on an interface is the variability in the different components that have to be connected. Detailed designs of interfaces related to variability obey the guideline provided by architectural textures.

8.3.1.2 Inputs

The following inputs should be available to perform the variability in the realization process:

— domain architecture including texture;

— domain implementation strategy;

— domain realization specification; and

— variability model in architecture.

8.3.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the variability in the realization process:

— *Variability model* is refined and extended.

— *Variability in realization* is explicitly documented.

8.3.1.4 Tasks

The organization shall implement the following tasks with respect to the variability in realization process:

— *Model variability in realization*: describe variability by using separate models.

— *Maintain variability mechanisms in realization*: support the usage of mechanisms to implements variability in realization.

— *Document variability in realization*: specify variability introduced during interface and component realization in order to support correct selection.

8.3.2 Model variability in realization

The goal of this task is to refine and extend the variability model in realization, as variability is refined or newly added at the realization stage.

The method should support modeling variability in realization with the following capabilities:

— supporting the variability refinement and extension in the variability model;

— providing ways for discriminating variability in the variability model that is newly added, refined or extended in realization from variability introduced in the previous stages; and

— supporting the verification and validation of variability model.

A tool should support modeling variability in realization by allowing the user to do the following:

— perform variability model refinement and extension in realization stage;

— utilize notations for discriminating variability in variability model that is newly introduced, refined or extended in realization with variabilities defined from the previous stages (e.g. annotation, colored notation);

— perform the maintenance of variability model configuration by each domain engineering process; and

— document variability model refinement history.

8.3.3 Maintain variability mechanisms in realization

The goal of this task is to find possible binding mechanisms for variability in realization.

The method should support maintaining variability mechanisms in realization with the following capabilities:

— defining variability mechanisms used in the realization stage (e.g. generic class, container class, parameterized abstract class);

— defining aspects to be checked for choosing the best variability mechanism in realization; and

— supporting decision for variability mechanisms.

A tool should support maintaining variability mechanisms in realization by allowing the user to do the following:

— perform simulation for the possible variability mechanism in realization;

— document the chosen variability mechanism; and

— record rationales for the decision.

8.3.4 Document variability in realization

The goal of this task is to conduct a proper documentation for interfaces and components that include variability. This includes a description of used variability mechanisms and possible reuse contexts.

The method should support documenting variability in interfaces with the following capabilities:

— confirming whether interfaces and components that include variability are consistent with variability model; and

— providing elaborations that differentiate variable interfaces and components from those in common [e.g. stereotype and note in UML models, XML (extensible markup language) markup tags].

A tool should support documenting variability in interfaces by allowing the user to do the following:

— utilize notations that differentiate variable interfaces and components from those in common (e.g. UML extensions, variability mechanism expressed by annotations to UML);

— visualize relationships among variable interfaces documented in different ways; and

— differentiate realization artefacts-related variability from those in common (e.g. parameterization tags in class diagram, stereotyped or tagged entity model, marked code fragments).

8.4 Traceability of variability in realization

8.4.1 Principal constituents

8.4.1.1 Purpose

The purpose of this subprocess is to establish and maintain links between realization artefacts related to variability and variability models defined separately.

8.4.1.2 Inputs

The following inputs should be available to perform the traceability of variability in realization process:

— detailed domain and application design specifications;

— domain and application implementation artefacts; and

— variability models in detailed design.

8.4.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the traceability of variability in realization process:

— *Trace links among different realization artefacts that include variability* are established and maintained.

— *Trace links between realization artefacts and variability models* are established.

8.4.1.4 Tasks

The organization shall implement the following tasks with respect to the traceability of variability in realization process:

— *Define trace links among variability in different realization artefacts*: establish and maintain links within variability introduced in realization artefacts.

— *Define trace links between realization artefacts and variability model*: establish and maintain links between realization artefacts and the variability model.

8.4.2 Define trace links among variability in different realization artefacts

The goal of this task is to establish and maintain explicit trace links among variability that are defined in different realization artefacts.

The method should support defining trace links among variability in different realization artefacts with the following capability:

— maintaining links among variability spread out different realization artefacts.

A tool should support defining trace links among variability in different realization artefacts by allowing the user to do the following:

— visualize links among variability in different realization artefacts.

8.4.3 Define trace links between realization artefacts and variability model

The goal of this task is to establish and maintain explicit links between domain/application realization artefacts and the domain/application variability model.

The method should support defining trace links between realization artefacts and variability model with the following capability:

— maintaining links of variability model with variability in realization artefacts.

A tool should support defining trace links between realization artefacts and variability model by allowing the user to do the following:

— visualize trace links between realization artefacts and variability model;

— record links between variability in realization and variability model.

9 Asset management in realization

9.1 General

During realization, detailed design artefacts, tangible and executable components, components that include variability, application detailed design artefacts and components and interfaces are generally managed as assets. Domain realization artefacts shall be properly structured and managed in order to be reused by member products. Application realization artefacts shall be properly structured and managed in order to be used by each relevant member product. Asset management in the realization process includes the following five key subprocesses:

— *Detailed domain design artefacts as domain assets* structures and maintains detailed domain design artefacts.

— *Domain implementation artefacts as domain assets* structures and maintains domain implementation artefacts.

— *Detailed application design artefacts as application assets* structures and maintains detailed application detailed design artefacts.

— *Application implementation artefacts as application assets* structures and maintains application implementation artefacts.

— *Attached process for detailed domain design asset reuse* is to define the process that application engineers should adhere to for properly reusing detailed domain assets.

NOTE ISO/IEC 26555 provides the details of asset management from asset identification through asset evolution.

9.2 Detailed domain design artefacts as domain assets

9.2.1 Principal constituents

9.2.1.1 Purpose

The purpose of this subprocess is to elaborate detailed domain design models and specification so as to support easy and correct reuse in detailed application design.

9.2.1.2 Inputs

The following inputs should be available to perform the detailed domain design artefacts as domain assets process:

— detailed domain design models; and

— detailed domain design specifications.

9.2.1.3 Outcomes

The following outcome shall be available as a result of the successful implementation of the detailed domain design artefacts as domain assets process:

— *Configurations of detailed domain design artefacts* are established.

9.2.1.4 Tasks

The organization shall implement the following tasks with respect to the detailed domain design artefacts as domain assets process:

— *Identify detailed design artefacts managed as domain assets*: determine detailed domain design artefacts that will be managed in the asset repository.

— *Define configuration and annotation in detailed domain design*: structure detailed domain design assets so they are simple to find and use.

9.2.2 Identify detailed design artefacts managed as domain assets

The goal of this task is to identify detailed domain design artefacts that will be reused as domain assets including both common and variable artefacts.

The method should support identifying detailed design artefacts managed as domain assets with the following capabilities:

— selecting detailed design models that have reuse potential;

— evaluating domain artefacts (e.g. checklist or evaluation criteria for evaluating reusability); and

— establishing backward trace links with domain architectural elements.

A tool should support identifying detailed design artefacts managed as domain assets with the following capabilities:

— accessing detailed domain design artefacts that have reuse potential; and

— recording the list of detailed domain design artefacts as domain assets.

9.2.3 Define configuration and annotation in detailed domain design

The goal of this task is to define and develop the further structure of detailed domain design artefacts that will help with reuse at the application developments.

The method should support defining configuration and annotation in detailed domain design with the following capabilities:

— defining configuration of detailed domain design artefacts that help retrieve, update, delete or maintain traceability;

— identifying annotations necessity to reuse detailed domain design artefacts as domain assets at the application developments; and

— validating configuration and annotations for detailed domain design artefacts.

A tool should support defining configuration and annotation in detailed domain design with the following capabilities:

— accessing the existing information for defining configuration and annotation (the configuration includes trace links with the relevant domain architectural elements);

— accessing the detailed domain design artefacts; and

— using an editor for template definition.

9.3 Domain implementation artefacts as domain assets

9.3.1 Principal constituents

9.3.1.1 Purpose

The purpose of this subprocess is to elaborate domain implementation artefacts such as components and interfaces necessary to reuse them in application component development.

9.3.1.2 Inputs

The following inputs should be available to perform the domain implementation artefacts as domain assets process:

— domain interfaces; and

— domain components.

9.3.1.3 Outcomes

The following outcome shall be available as a result of the successful implementation of the domain implementation artefacts as domain assets process:

— *Configurations of domain implementation artefacts* are established.

9.3.1.4 Tasks

The organization shall implement the following tasks with respect to the domain implementation artefacts as domain assets process:

— *Identify domain implementation artefacts managed as domain assets*: determine domain components and domain interfaces that will be managed in asset repository.

— *Define configuration and annotation in domain implementation*: structure domain component and domain interface so as easy to find and use them.

9.3.2 Identify domain implementation artefacts managed as domain assets

The goal of this task is to identify domain components and domain interfaces that will be reused as domain assets including both common and variable artefacts.

The method should support identifying domain implementation artefacts managed as domain assets with the following capabilities:

— selecting components and interfaces that have reuse potential;

— evaluating components and interfaces (e.g. checklist or evaluation criteria for evaluating reusability); and

— establishing backward trace links with relevant detailed design artefacts.

A tool should support identifying domain implementation artefacts managed as domain assets with the following capabilities:

— accessing components and interfaces that have reuse potential; and

— recording the list of components and interfaces as domain assets.

9.3.3 Define configuration and annotation in domain implementation

The goal of this task is to define and develop the further structure of component and interface implementations to help reuse in the application developments.

The method should support defining configuration and annotation in domain implementation with the following capabilities:

— defining configuration of component and interface implementations that help retrieve, update, delete or maintain traceability;

— identifying annotations necessity to reuse component and interface implementations as domain assets at the application developments; and

— validating configuration and annotations for component and interface implementations.

A tool should support defining configuration and annotation in domain implementation with the following capabilities:

— accessing the existing information for defining configuration and annotation (the configuration includes trace links with the relevant detailed domain design artefacts);

— accessing information related to components or interfaces; and

— using an editor for the template definition.

9.4 Attached process for reusing domain realization assets

9.4.1 Principal constituents

9.4.1.1 Purpose

The purpose of this subprocess is to define attached process that will be adhered for the correct reuse of domain realization assets.

9.4.1.2 Inputs

The following inputs should be available to perform the attached process for reusing domain realization assets process:

— domain realization assets.

9.4.1.3 Outcomes

The following outcome shall be available as a result of the successful implementation of the attached process for reusing domain realization assets process:

— *Domain realization assets including attached process* are established.

9.4.1.4 Tasks

The organization shall implement the following tasks with respect to the attached process for reusing domain realization assets process:

— *Identify processes adhered for realization asset reuse*: determine processes to be attached to domain realization assets.

— *Make attached process as a part of domain realization assets*: add the identified processes to relevant assets so as a member product follows them for domain asset reuse.

9.4.2 Identify processes adhered for realization asset reuse

The goal of this task is to identify and define processes for which application realization should follow for the correct reuse of domain realization assets.

The method should support identifying processes adhered for realization asset reuse with the following capabilities:

— identifying domain assets of which process standard is important;

— defining attached process based on domain engineering processes;

— assessing attached process; and

— providing documentation templates.

A tool should support identifying processes adhered for realization asset reuse with the following capabilities:

— accessing information for domain realization assets;

— referring organizational standard domain engineering processes; and

— documenting the attached process.

9.4.3 Make attached process as a part of domain realization assets

The goal of this task is to integrate attached process with related domain assets, so that application engineers can access together.

The method should support making attached process as a part of domain assets with the following capabilities:

— providing glues to integrate attached processes into domain realization assets; and

— attaching processes to each related domain realization asset.

A tool should support making attached process as a part of domain assets with the following capabilities:

— providing ways to attach processes to domain realization assets; and

— allowing the automatic access of attached processes when a domain realization asset is reused.

9.5 Detailed application design artefacts as application assets

9.5.1 Principal constituents

9.5.1.1 Purpose

The purpose of this subprocess is to elaborate detailed application design artefacts necessary for reuse in each application development.

9.5.1.2 Inputs

The following inputs should be available to perform the detailed application design artefacts as application assets process:

— detailed designs of application interfaces; and

— detailed designs of application components.

9.5.1.3 Outcomes

The following outcome shall be available as a result of the successful implementation of the detailed application design artefacts as application assets process:

— *Configurations of application interfaces and components* are established.

9.5.1.4 Tasks

The organization shall implement the following tasks with respect to the detailed application design artefacts as application assets process:

— *Identify detailed application design artefacts managed as application assets*: determine application design artefacts that will be managed in asset repository.

— *Define configuration and annotation in detailed application design*: structure detailed application design so that it is easy to find and use.

9.5.2 Identify detailed application design artefacts managed as application assets

The goal of this task is to identify detailed application design artefacts such as models and specifications to be maintained in each application development.

The method should support identifying detailed application design artefacts managed as application assets with the following capability:

— selecting detailed application designs artefacts used at the later processes of application development.

A tool should support identifying detailed application design artefacts managed as application assets with the following capabilities:

— importing detailed application design artefacts; and

— performing reviewing and editing application detailed design artefacts.

9.5.3 Define configuration and annotation in detailed application design

The goal of this task is to define or develop the structure and information for detailed application design assets that will be referred to by the successive application development. Configuration for detailed application design assets that will help them to be managed as assets are defined.

The method should support defining configuration and annotation in detailed application design assets with the following capabilities:

— defining configuration of detailed application design assets that help retrieve, update, delete or maintain traceability;

— identifying annotations necessity to use detailed application design assets at the application developments; and

— validating configuration and annotations for detailed application design assets.

A tool should support defining configuration and annotation in detailed application design assets with the following capabilities:

— accessing the existing information for defining configuration and annotation;

— accessing the detailed application design assets; and

— using an editor for template definition.

9.6 Application implementation artefacts as application assets

9.6.1 Principal constituents

9.6.1.1 Purpose

The purpose of this sub process is to elaborate application components necessary for reuse in each application development.

9.6.1.2 Inputs

The following inputs should be available to perform the application implementation artefacts as application assets process:

— implementation artefacts, such as application components and application interfaces.

9.6.1.3 Outcomes

The following outcome shall be available as a result of the successful implementation of the application implementation artefacts as application assets process:

— *Configurations of application implementation artefacts* are established.

9.6.1.4 Tasks

The organization shall implement the following tasks with respect to the application implementation artefacts as part of the application assets process:

— *Identify application implementation artefacts as application assets*: determine application components and interfaces that will be managed in asset repository.

— *Define configuration and annotation of application implementation*: structure application components and interfaces so that they are easy to find and use.

9.6.2 Identify application implementation artefacts as application assets

The goal of this task is to identify application components and interfaces to be maintained in each application development.

The method should support identifying application implementation artefacts managed as application assets with the following capability:

— selecting application implementation artefacts used at the later processes of application development.

A tool should support identifying application implementation artefacts managed as application assets with the following capabilities:

— importing application implementation artefacts; and

— performing reviewing and editing application implementation artefacts.

9.6.3 Define configuration and annotation of application implementation

The goal of this task is to define or develop the structure and information for application components that will be referred to by the successive application development. Configuration for application components that will help it to be managed as assets are defined.

The method should support defining configuration and annotation in application implementation with the following capabilities:

— defining configuration of application components that help retrieve, update, delete or maintain traceability;

— identifying annotations necessity to use application components at the application developments; and

— validating configuration and annotations for application components.

A tool should support defining configuration and annotation in application implementation with the following capabilities:

— accessing the existing information for defining configuration and annotation;

— accessing the application implementation assets; and

— using an editor for the template definition.

10 Detailed application design

10.1 General

Detailed application design reuses common design artefacts, selects variants of which the binding time of variability is detailed design stage, and conducts detailed design for application interfaces and components. Detailed application design can add detailed designs to the detailed application design configuration derived from domain design. Detailed application design process includes the following three key sub- processes:

— *Detailed application design initiation* derives configuration for a member product through variability binding in detailed design in order to start detailed application design.

— *Detailed application interface design* conducts low-level designs for application interfaces.

— *Detailed application component design* conducts low-level designs for application components.

10.2 Detailed application design initiation

10.2.1 Principal constituents

10.2.1.1 Purpose

The purpose of this subprocess is to derive an application configuration by reusing common interfaces/components and by binding variable interfaces/components delivered by domain design and implementation so as to initiate detailed application design from thereafter. For starting detailed application design, detailed design configurations are derived from detailed domain design and application architecture should be ready for guiding detailed application design.

10.2.1.2 Inputs

The following inputs should be available to perform the detailed application design initiation process:

— detailed domain design specification; and

— application architecture design specification.

10.2.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the detailed application design initiation process:

— *Detailed application design specification* is derived from detailed domain design specification.

— *Detailed application design capability* is established.

— *Detailed application design* is initiated.

10.2.1.4 Tasks

The organization shall implement the following tasks with respect to the detailed application design initiation process:

— *Derive detailed application design from detailed domain design*: configure detailed application design in accordance with application architecture and binding decisions in detailed design.

— *Validate derived detailed application design*: help ensure the derived detailed application design conforms to the application architecture.

— *Confirm application implementation capability*: check whether required application implementation capabilities are established.

10.2.2 Derive detailed application design from detailed domain design

The goal of this task is to apply bindings and to produce initial detailed application design including models and specifications.

The method should support deriving detailed application design from detailed domain design with the following capabilities:

— providing binding information (e.g. binding time, binding mechanism) related to a member product;

— extracting relevant configuration of detailed application design specification from detailed domain design specification; and

— providing integrated description language for detailed domain/application component and interface.

A tool should support deriving detailed application design from detailed domain design by allowing the user to do the following:

— store and access binding information related to a member product;

— extract detailed application designs and specification automatically from detailed domain design; and

— describe integrated domain/application component and interface.

10.2.3 Validate derived detailed application design

The goal of this task is to help ensure that the derived detailed application designs conform to application requirements and application architecture.

The method should support validating derived detailed application designs with the following capabilities:

— defining evaluation criteria or checklists; and

— tracing forward relationships among architecture and detailed design artefacts.

A tool should support validating derived application design specification by allowing the user to do the following:

— evaluate application design step by step in accordance with evaluation criteria or checklists;

— store forward relationships among architecture and detailed design artefacts; and

— record and report evaluation results.

10.2.4 Confirm detailed application design capability

The goal of this task is to check whether detailed application design capability is ready to initiate detailed application design immediately.

The method should support confirming detailed application design capability with the following capability:

— providing confirmation aspects for ensuring the readiness(e.g., plans, procedures, standards, textures, tools, resources) of application implementation.

A tool should support confirming detailed application design capability by allowing the user to do the following:

— access information for the established organizational level enabling environments.

10.3 Detailed application interface design

10.3.1 Principal constituents

10.3.1.1 Purpose

The purpose of this subprocess is to define detailed design of application interfaces that cannot be realized by reusing interfaces delivered by domain realization.

10.3.1.2 Inputs

The following inputs should be available to perform the detailed application interface design process:

— application configuration;

— application requirements specification; and

— application architecture specification.

10.3.1.3 Outcomes

The following outcome shall be available as a result of the successful implementation of the detailed application interface design process:

— *Detailed design specification for a member product including application interface* is produced.

10.3.1.4 Tasks

The organization shall implement the following tasks with respect to the detailed application interface design process:

— *Examine interactions among application components*: analyse the configuration parameters and the return values of interactions among application components.

— *Define the detailed internal structures of application interfaces*: design methods provided or required by application interfaces.

— *Verify and validate detailed application interface design*: help ensure that the noncompliance issues against application architecture and errors in detailed application interface design are identified and resolved.

— *Document detailed application interface design*: specify the detailed design of application interfaces.

10.3.2 Examine interactions among application components

The goal of this task is to capture the interactions among components which will be used to design the detailed structure of application interfaces.

The method should support examining interactions among application components with the following capabilities:

— looking for the interactions among components consisting application architecture; and

— defining services provided or required for the interactions.

A tool should support examining interactions among application components by allowing the user to do the following:

— access application architecture design; and

— visualize interactions between application components.

10.3.3 Define the detailed internal structures of application interfaces

The goal of this task is to define the methods with their configuration parameters of application interfaces.

The method should support defining the detailed internal structures of application interfaces with the following capability:

— harmonizing application interfaces with common and variable domain interfaces.

A tool should support defining the detailed internal structures of application interfaces by allowing the user to do the following:

— browse the detailed design models for commonality and bound variability; and

— model application interfaces under the integration of commonality and bound variability.

10.3.4 Verify and validate detailed application interface design

The goal of this task is to confirm whether the detailed application interface design adheres to the rules defined in application architecture specification and help ensure the correctness of the detailed application interface design.

The method should support verifying and validating the detailed design of application interfaces with the following capabilities:

— selecting test requirements, test cases and test procedures designed during domain interface verification and validation to be re-executed;

— designing test requirements, test cases and test procedures for application-specific interfaces; and

— supporting validation for conformances among application interfaces and reused domain interfaces.

A tool should support verifying and validating the detailed design of application interfaces by allowing the user to do the following:

— access application architecture specification;

— access application-specific requirements specification; and

— send feedback for the identified non-conformance issues to related domain engineering or application engineering process.

10.3.5 Document detailed application interface design

The goal of this task is to make detailed application interface design specification.

The method should support documenting detailed application interface design with the following capability:

— harmonizing application interface documentation with the derived document.

A tool should support documenting detailed application interface design by allowing the user to do the following:

— access the derived document; and

— import the detailed design of application interfaces for documentation.

10.4 Detailed application component design

10.4.1 Principal constituents

10.4.1.1 Purpose

The purpose of this subprocess is to define detailed design of application components that cannot be realized by reusing components delivered by domain realization.

10.4.1.2 Inputs

The following inputs should be available to perform the detailed application component design process:

— application configuration;

— application requirements;

— application architecture specification; and

— detailed application component and interfaces design specification.

10.4.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the detailed application component design process:

— *Evaluation results for detailed application design* are documented.

— *Detailed application design specification including application components* is produced.

10.4.1.4 Tasks

The organization shall implement the following tasks with respect to the detailed application component design process:

— *Identify, evaluate, and select COTS*: identify and evaluate COTS components that meet member product specific requirements.

— *Define the detailed internal structures of application components*: design the internal structures of an application component.

— *Verify and validate detailed application component design*: help ensure that noncompliance issues against application architecture, errors in detailed application component designs and in-conformances against related application interfaces are identified and resolved.

— *Document the detailed design of application components*: specify the detailed application component design.

10.4.2 Identify, evaluate and select COTS

The goal of this task is to find appropriate components with low costs and improve the quality of the member product.

The method should support identifying, evaluating and selecting COTS with the following capabilities:

— defining evaluation criteria for member product-specific COTS selection;

— supporting viable COTS selection;

— supporting cost estimation necessary for tailoring or modifying COTS for a member product; and

— supporting COTS adaption for integration with application components if necessary.

A tool should support identifying, evaluating and selecting COTS by allowing the user to do the following:

— access information for the candidate COTS component;

— perform data collection and calculation for cost estimation; and

— record rationale for decision.

10.4.3 Define the detailed internal structures of application components

The goal of this task is to design application components in accordance with the services defined in application interfaces.

The method should support defining the detailed internal structures of application components with the following capabilities:

— expressing explicitly application components in a way that differentiates them from derived components; and

— visualizing application components.

A tool should support defining the detailed internal structures of application components by allowing the user to do the following:

— utilize explicit notation for expressing application components.