INTERNATIONAL STANDARD

ISO/IEC 26550

Second edition
2015-12-01

Software and systems engineering — Reference model for product line engineering and management

Ingénierie du logiciel et des systèmes - Modèle de référence pour l'ingénierie et la gestion de lignes de produits

Reference number
ISO/IEC 26550:2015(E)

COPYRIGHT PROTECTED DOCUMENT

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

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 on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword - Supplementary information

The committee responsible for this document is ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*.

This second edition cancels and replaces the first edition (ISO/IEC 26550:2013), of which it constitutes a minor revision.

Introduction

Software and Systems Product Line (SSPL) engineering and management creates, exploits, and manages a common platform to develop a family of products (e.g. software products, systems architectures) at lower cost, reduced time to market, and with better quality. As a result, it has gained increasing global attention since 1990s.

This International Standard provides a reference model consisting of an abstract representation of the key processes of software and systems product line engineering and management and the relationships between the processes. Two key characteristics, the need for both domain and application engineering lifecycle processes and the need for the explicit variability definition, differentiate product line engineering from single-system engineering. The goal of domain engineering is to define and implement domain assets commonly used by member products within a product line, while the goal of application engineering is to develop applications by exploiting the domain assets including common and variable assets. Domain engineering explicitly defines product line variability which reflects the specific needs of different markets and market segments. Variability may be embedded in domain assets. During application engineering, the domain assets are deployed in accordance with the defined variability models.

The reference model for SSPL engineering and management can be used in subsequent standardization efforts to create standards having a high level of abstraction (e.g. product management, scoping, requirements engineering, design, realization, verification and validation, and organizational and technical management), a medium level of abstraction (e.g. configuration management, variability modeling, risk management, quality assurance, measurement, evaluation, asset repository), or a detailed level of abstraction (e.g. texture, configuration mechanism, asset mining) for software and systems product line engineering.

Software and systems engineering — Reference model for product line engineering and management

1 Scope

This International Standard is the entry point of the whole suite of International Standards for software and systems product line engineering and management.

The scope of this International Standard is to

— provide the terms and definitions specific to software and systems product line engineering and management,

— define a reference model for the overall structure and processes of software and systems product line engineering and management and describe how the components of the product line reference model fit together, and

— define interrelationships between the components of the product line reference model.

This International Standard does not describe any methods and tools associated with software and systems product line engineering and management. Descriptions of such methods and tools will appear in the consecutive International Standards (ISO/IEC 26551[1)] to ISO/IEC 26556[2)]). This International Standard does not deal with terms and definitions addressed by ISO/IEC/IEEE 24765:2010 that provides a common vocabulary applicable to all systems and software engineering work.

Whenever this International Standard refers to "products", it means "system-level products" consisting of software systems or both hardware and software systems. It may be useful for the engineering and management of product lines that consist of only hardware systems but it has not been explicitly created to support such hardware product lines. This International Standard is not intended to help the engineering, production, warehousing, logistics, and management of physical items that, possibly combined with software, comprise the products. These processes belong to other disciplines (e.g. mechanics, electronics).

NOTE Annex A provides further information on products.

This International Standard, including the product line reference model and the terms and definitions, has been produced starting from References [6], [7], and [8] which finally resulted in a broad consensus from National Member Bodies at the time of publication. In addition to this background process, structures from ISO/IEC 12207:2008, ISO/IEC/IEEE 15288:2015, ISO/IEC 15940:2006 and ISO/IEC 14102:2008 have been used as a baseline.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

There are no normative references cited in this document.

1) Second edition to be published.

2) Under development.

3 Terms and definitions

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

3.1
application architecture
architecture including the architectural structure and rules (e.g. common rules and constraints) that constrains a specific member product within a product line

Note 1 to entry: The application architecture captures the high-level design of a specific member product of a product line. An application architecture of the member products included in the product line reuses (possibly with modifications) the common parts and binds variable parts of the domain architecture. In most cases, an application architecture of the member products needs to develop application-specific variability.

3.2
application asset
output of a specific application engineering process (e.g. application realization) that may be exploited in other lifecycle processes of application engineering and may be adapted as a domain asset based on a product management decision

Note 1 to entry: Application asset encompasses requirements, an architectural design, components, and tests. In contrast to domain assets that need to support the mass-customization of multiple applications within the product line, most application assets do not contain variability. However, applications may possess variability (e.g. end-users may be enabled to mass-customize the applications they are using by binding application variability during run time). Application Assets may thus possess variability as well, but the variability of an application asset only serves the purposes of the particular application, for which the application asset has been created. As a result, the scope of application asset variability is typically much narrower than the scope of domain asset variability.

Note 2 to entry: Application assets are not physical products available off-the-shelf and ready for commissioning. Physical products (e.g. mechanical parts, electronic components, harnesses, optic lenses) are stored and managed according to the best practices of their respective disciplines. Application assets have their own life cycles; ISO/IEC/IEEE 15288 may be used to manage a life cycle.

3.3
application design
process of application engineering where a single application architecture conforming to the domain architecture is derived

3.4
application engineering
life cycle consisting of a set of processes in which the application assets and member products of the product line are implemented and managed by reusing domain assets in conformance to the domain architecture and by binding the variability of the platform

Note 1 to entry: Application engineering in the traditional sense means the development of single products without the strategic reuse of domain assets and without explicit variability modeling and binding.

3.5
application realization
process of application engineering that develops application assets, some of which may be derived from domain assets, and member products based on the application architecture and the sets of application assets and domain assets

3.6
asset base
reusable assets produced from both domain and application engineering

3.7
asset scoping
process of identifying the potential domain assets and estimating the returns of investments in the assets

Note 1 to entry: Information produced during asset scoping, together with the information produced by product scoping and domain scoping, can be used to determine whether to introduce a product line into an organization. Asset scoping takes place after domain scoping.

3.8
binding
task to make a decision on relevant variants, which will be application assets, from domain assets using the domain variability model and from application assets using the application variability model

Note 1 to entry: Performing the binding is a task to apply the binding definition to generate new application from domain and application assets using the domain and application variability models.

3.9
commonality
set of functional and non-functional characteristics that is shared by all applications belonging to the product line

3.10
domain architecture
reference architecture
product line architecture
core architecture that captures the high-level design of a software and systems product line including the architectural structure and texture (e.g. common rules and constraints) that constrains all member products within a software and systems product line

Note 1 to entry: Application architectures of the member products included in the product line reuse (possibly with modifications) the common parts and bind variable parts of the domain architecture. Application architectures of the member products may (but do not need to) provide variability.

3.11
domain asset
core asset
output of domain engineering life cycle processes and can be reused in producing products during application engineering

Note 1 to entry: Domain assets may include domain features, domain models, domain requirements specifications, domain architectures, domain components, domain test cases, domain process descriptions, and other assets.

Note 2 to entry: In systems engineering, domain assets may be subsystems or components to be reused in further system designs. Domain assets are considered through their original requirements and technical characteristics. Domain assets include but are not limited to use cases, logical principles, environmental behavioral data, and risks or opportunities learnt from previous projects. Domain assets are not physical products available off-the-shelf and ready for commissioning. Physical products (e.g. mechanical parts, electronic components, harnesses, optic lenses) are stored and managed according to the best practices of their respective disciplines. Domain assets have their own life cycles. ISO/IEC/IEEE 15288 may be used to manage a life cycle.

3.12
domain engineering
life cycle consisting of a set of processes for specifying and managing the commonality and variability of a product line

Note 1 to entry: Domain assets are developed and managed in domain engineering processes and are reused in application engineering processes.

Note 2 to entry: Depending on the type of the domain asset, that is, a system domain asset or a software domain asset, the engineering processes to be used may be determined by the relevant discipline.

Note 3 to entry: IEEE 1517-2010, Clause 3 defines domain engineering as a reuse-based approach to defining the scope (i.e., domain definition), specifying the structure (i.e., domain architecture), and building the assets (e.g. requirements, designs, software code, documentation) for a class of systems, subsystems, or member products.

3.13
domain scoping
subprocess for identifying and bounding the functional domains that are important to an envisioned product line and provide sufficient reuse potential to justify the product line creation

3.14
feature
abstract functional characteristic of a system of interest that end-users and other stakeholders can understand

Note 1 to entry: In systems engineering, features are syntheses of the needs of stakeholders. These features will be used, amongst others, to build the technical requirement baselines.

3.15
member product
application
product belonging to the product line

3.16
product line
product family
set of products and/or services sharing explicitly defined and managed common and variable features and relying on the same domain architecture to meet the common and variable needs of specific markets

3.17
product line architecture
synonym of domain architecture

3.18
product line platform
product line architecture, a configuration management plan, and domain assets enabling application engineering to effectively reuse and produce a set of derivative products

Note 1 to entry: Platforms have their own life cycles. ISO/IEC/IEEE 15288 may be used to manage a life cycle.

3.19
product line reference model
abstract representation of the domain and application engineering life cycle processes, the roles and relationships of the processes, and the assets produced, managed, and used during product line engineering and management

3.20
product line scoping
process for defining the member products that will be produced within a product line and the major common and variable features among the products, analyzes the products from an economic point of view, and controls and schedules the development, production, and marketing of the product line and its products

Note 1 to entry: Product management is primarily responsible for product line scoping.

3.21
product scoping
subprocess of product line scoping that determines the product roadmap, that is (1) the targeted markets; (2) the product categories that the product line organization should be developing, producing, marketing, and selling; (3) the common and variable features that the products should provide in order to reach the long and short term business objectives of the product line organization, and (4) the schedule for introducing products to markets

3.23
variability
characteristics that may differ among members of the product line

Note 1 to entry: The differences between members may be captured from multiple viewpoints such as functionality, quality attributes, environments in which the members are used, users, constraints, and internal mechanisms that realize functionality and quality attributes.

Note 2 to entry: It is important to distinguish between the concepts of system and software variability and product line variability. Any system partially or fully composed of software can be considered to possess software variability because software systems are inherently malleable, extendable, or configurable for specific use contexts. Product line variability is concerned with the variability that is explicitly defined by product management. This International Standard is primarily concerned with product line variability.

3.24
variability constraint
constraint relationships between a variant and a variation point, between two variants, and between two variation points

3.25
variability dependency
relationship between a variation point and a set of variants, which indicates that the variation point implies a decision about the variants

Note 1 to entry: Two kinds of variability dependencies are possible: (1) the optional variability dependency states that the variant optionally dependent on a variation point can be a part of a member product of a product line; (2) the mandatory variability dependency defines that a variant dependent on a variation point must be selected for a member product if the variation point is selected for the member product.

3.26
variability management
managerial tasks relate to variability and has two dimensions: variability dimension and asset dimension

Note 1 to entry: Variability management in the variability dimension consists of tasks for overseeing variability in the level of the entire product line, creating and maintaining variability models, ensuring consistencies between variability models, managing all variability and constraint dependencies across the product line, and managing the traceability links between a variability model and associated domain and application assets (e.g. requirements models, design models). Variability management in the asset dimension consists of tasks for managing the impacts of variability within each domain and application asset, that is, in which location of an asset a particular variability occurs and which alternative shapes the asset can take in that location. The dimensions are complementary in nature, that is, both are needed for successful variability management.

3.27
variability model
explicit definition for product line variability

Note 1 to entry: It introduces variation points, types of variation for the variation points, variants offered by the variation points, variability dependencies, and variability constraints. Variability models may be orthogonal to or integrated in other models such as requirements or design models. There are two types of variability models: application variability models and domain variability models.

3.28
variant
one alternative that may be used to realize particular variation points

Note 1 to entry: One or more variants must correspond to each variation point. Each variant has to be associated with one or more variation points. Selection and binding of variants for a specific product determine the characteristics of the particular variability for the product.

3.29
variation point
representation corresponding to particular variable characteristics of products, domain assets, and application assets in the context of a product line

Note 1 to entry: Variation points show what of the product line varies. Each variation point should have at least one variant.

4 From single-system engineering and management toward product line engineering and management

Single-system engineering and management is the dominant way of conceptualizing and developing software and systems products. This Clause first outlines some of the main challenges software and systems product companies face in using single-system engineering and management approaches. It identifies variability management as the most challenging area. Variability management is discussed in 4.1. The clause concludes by explaining major differences between single-system engineering and management and product line engineering and management. Understanding these differences is a key for successful organizational transitioning from single-system engineering and management toward product line engineering and management.

4.1 Challenges product companies face in the use of single-system engineering and management

The excessive use of single-system engineering and management in environments where the assumptions no longer hold contributes to a variety of issues encountered by customers, end-users, and providers. For example, customers may feel their needs are unique and acquire and sustain expensive tailored systems while commoditized, inexpensive products might be completely adequate. End-users may experience that the functionality they really need is difficult to find and/or use because the software systems are too complex and provide too much functionality. Finally, a provider may sell several interrelated products, which look and feel completely different and do not interoperate, even to the same customers.

Providers of single products typically encounter at least some of the following issues when using single-system engineering: work efforts and costs are underestimated, productivity is overestimated, must-have features are missing, product schedules and/or quality goals are not met, and/or customer satisfaction remains lower than expected. Work efforts may be underestimated and productivity overestimated because the organization has never before created a similar product or if it has, the organizational unit who created the similar product may not want to share its experiences and other possibly reusable assets due to rivalry between organizational units. Inaccurate estimates together with typically fixed budgets result in schedule fluctuations, missing features, and/or quality issues. Quality issues may also result from the lack of a reuse culture because the software developed from scratch typically has much higher defect density than the software reusing well-tested components.

The accommodation of adequate variability is typically the most significant problem faced by the providers of single products. In this context, the variability needs typically emerge over time from interactions with various customers. Providers commonly use one or more seemingly simple but ineffective tactics to deal with emerging variability. For example, a provider may incorporate variability into a single product by introducing more and more (partly end-user-visible) parameters in the product and more and more if-then-else-statements in the source code text of the product to deal with the parameters during run time. As a result, the number of source code lines grows, the source code becomes increasingly complex to understand and maintain, and the testability (and often also the performance) of the software deteriorates. Alternatively, a provider with an existing product may deal with the variable requirements of a new customer by branching a new product from the existing product, modifying the source code of the new product, merging the modified source code back to the main line when there is time and other resources available, and finally deleting the branch. Branching and merging is very expensive and error prone and the source code of the main line will typically become very complex after a few branches and merges, requiring expensive periodical refactoring to utilize the tactic on a long term basis. In the worst case, the provider may end up with many partially cloned

products and no main line of source code to maintain. Such ineffective tactics for managing variability also make the jobs of software developers tedious and are likely to increase employee turnover.

In sum, product companies utilizing single-system engineering and management approaches may end up with highly complex and low-quality products, low productivity, high employee turnover, and less than expected customer satisfaction.

Product line engineering and management is a possible way of dealing with such problems. However, it is no panacea. If it is understood and implemented poorly, significant investments may result without the materialization of expected benefits. Therefore, the following clauses of this International Standard outline what product line engineering and management is and how providers can leverage it to establish and manage variability; reduce costs and product complexity; increase productivity and product quality through strategic, prescribed creation and use of domain assets; shorten time to market; and increase customer satisfaction through mass-customization of products and more accurate estimation of schedules and costs.

4.2 Variability management

In single-system engineering and management, reuse of knowledge is important. However, product line engineering and management fundamentally differs from single-system engineering and management. Product line engineering and management has to take explicitly into account multiple products and the variations within and between them. Some variability needs can still emerge (e.g. based on unexpected offerings of competitors) because perfect upfront planning of variability is impossible but most variability needs should be based on the careful analysis of target markets, available technologies, offerings of competitors, and other factors. Distinction between common and variable parts of members of the product line affects product line engineering and management in many ways. Some examples are provided next.

— Developing the domain architecture: Common and variable parts of products in the product line must be clearly distinguished in the domain architecture of the product line.

— Ensuring traceability: Variability within and between members of the product line is located in various domain and application assets, including variability models. Domain and application assets must be traced, respectively, throughout the domain and application engineering life cycles. Because application assets may reuse domain assets as they are or after modifications, application assets must also be bidirectionally traceable to domain assets. As a result, traceability is also necessary across the domain and application engineering life cycles.

Variability is thus a key differentiator between single-system engineering and management and product line engineering and management. Variability must be defined, modeled, implemented, versioned, verified and validated. It must also be traced within and across domain and application engineering life cycles. The discipline for managing variability is called variability management. The most central concepts of product line engineering and management, discussed in 4.3, are strongly related to variability management.

4.3 Key differentiators between single-system engineering and management and product line engineering and management

The identification and analysis of key differentiators between single-system engineering and management and product line engineering and management will help organizations to understand the product line reference model (5.2) and to formulate a strategy for successful implementation of product line engineering and management. Product line organizations should thus design their structures and processes to address these differentiators.

— Application engineering: A process life cycle in which application assets and member products of a product line are implemented and managed by reusing domain assets in conformance to the domain architecture and by binding the variability of the product line. Thus, the existence of two life cycle processes, that is, domain engineering and application engineering, distinguishes product line engineering from single-system engineering.

— Binding: A decision making task that distinguishes product line engineering from single-system engineering. It resolves a variety of optional or alternative behaviors provided by domain assets and application assets and represented by variability models to create application assets or member products. Binding should be considered during domain engineering at a time when the variants are introduced, as well as during and after application engineering at the time when the variants are bound. Static binding of variability during application engineering takes place before run time. Dynamic binding of variability can be used during run time. It enables a system (1) to self-adapt its behavior based on pre-specified rules and (2) to be adapted by its users (e.g. when they decide to bind additional variants to use enhanced features).

— Collaboration: Because there are two parallel life cycles in product line engineering, people responsible for domain engineering processes need to collaborate within domain engineering and also with people in related application engineering processes.

— Configuration management: Configurations of a product line are multidimensional in time and space. Application and domain assets, platform releases, and member products have versions. Each of these versions has a configuration. Versions of member products depend on asset versions and platform release versions. Platform release versions depend on domain asset versions. For example, changes in domain assets may impact numerous member products. Each member product can exist in multiple configurations at any given time. The possible configurations of a member product can change over time. The dimensions need to be managed. While configuration management is necessary for all systems incorporating software, it is thus of paramount importance to product line engineering and management.

— Domain architecture: It captures the high-level design of a product line, including the variability defined by domain engineering. It is used as a blueprint for designing the architecture and texture of all product line members. The need for domain architectures (or reference architectures) distinguishes product line engineering from single-system engineering.

— Domain asset: The existence of domain assets in software and systems product lines is another differentiator between product line engineering and single-system engineering.

— Domain engineering: It defines, realizes, verifies and validates the domain assets. Domain variability model of the product line is structured throughout this process life cycle. It distinguishes product line engineering from single-system engineering. For each relevant process within the life cycle, the required subprocesses, roles, and procedures should be described.

— Enabling technology support: Technologies needed to enable product line engineering and management should be available for the successful product line implementation. Knowledge management infrastructure is especially crucial because software product line engineering and management is more knowledge intensive than single-system engineering. For example, product lines typically consist of more assets, more different types of assets, and more dependencies between assets than individual systems. Managing knowledge about valid asset configurations is thus more challenging in product line engineering than in single-system engineering.

— Measurement and tracking: The measurement involved in product line engineering and management is multidimensional. Data collection, measures, and tracking need to support the domain engineering and application engineering life cycles, as well as organizational and technical management of product lines, in a balanced way. For example, inadequate or biased measurement of the organizational units responsible for application engineering and domain engineering is likely to lead to disputes and misallocations of funding and other resources between the organizational units. Such disputes may deteriorate the funding of domain engineering because most, if not all, organizational revenue is typically obtained from the sales of member products and related services, not from the sales of domain assets. The lack of a fair funding model and supporting measurement and tracking systems may hamper or even destroy the organizational implementation and institutionalization of product line engineering and management.

— Platform: Product line engineering is platform-based. Mass-customization of products is practically impossible without effective platforms. Platforms thus distinguish product line engineering from

single-system engineering and are of strategic importance for product line organizations. The introduction and elimination of entire platforms influence product line organizations significantly.

— Product management: It is responsible for the economic and business concerns of product line engineering and management and the resulting product line(s). It deals particularly with the market strategy and the competitive strategy. Product lines must evolve continuously in accordance with new innovative products and application assets that products within the product line can leverage, market changes, and new offerings from competitors. Good practices for product line engineering and management include evolving a product line in iterative cycles, establishing clear objectives for each cycle, and reviewing performance after completing each cycle. Product management is responsible for these practices and for making appropriate adjustments to the product portfolio and the platform investments based on the review results. Product management is needed in single-system engineering and management but it plays a more vital and powerful role in product line engineering and management.

— Reusability: The reusability of domain assets is one of the critical success factors of product lines. Reusability requires strategic long-term focus on key domains, so investments in developing domain assets are feasible, and sound technological and managerial capabilities throughout the product line engineering and management processes.

— Texture: Architectural texture contains common rules and constraints (e.g. common architectural styles or design patterns for specific solutions, common glues facilitating the composition of architectural components) for the design and realization of all member products and domain and application assets in the product line. While texture is important in single-system engineering, it has a key role in product line engineering because the texture of the domain architecture governs the architectures of numerous member products (e.g. to ensure the common look and feel of the member products within a product line).

— Traceability: Product line engineering and management is typically more knowledge-intensive than single-system engineering and management. Product line organizations need to manage knowledge related to not only more assets but also to far more associations between assets. For example, variability models may be bidirectionally associated with domain and application assets and domain assets may be bidirectionally associated with application assets. Product line organizations thus need to develop and maintain a knowledge management infrastructure that also covers the management of such traceability links.

— Verification and validation: They confirm through the provision of objective evidence that the requirements for all domain assets and member products are fulfilled. For example, the verification and validation of a domain asset provide confidence, respectively, that the asset performs the specified services with an agreed upon level of quality and helps the system (e.g. a member product) requiring the services of the asset to achieve its goals. Verification and validation in product line context are fundamentally different from the single-system engineering context.

— Variability: The product line variability defined by product management and enabled by the product line platform allows flexible and effective mass-customization of member products within a product line, distinguishing product line engineering and management from single-system engineering and management.

5 Reference model for product line engineering and management

This Clause defines a reference model that forms the basis for designing and executing product line engineering processes and enabling capabilities of methods and tools. The product line reference model defines, at a high level, the fundamental elements of product line engineering and management and their relationships.

5.1 General

The product line reference model addresses both engineering and management processes and covers the key characteristics of product line engineering and management.

The product line reference model provides an overview of the consecutive International Standards (i.e., ISO/IEC 26551 to ISO/IEC 26556), as well as the structure of the model.

— Processes and capabilities of methods and tools for product line scoping, domain requirements engineering, and application requirements engineering are described by ISO/IEC 26551.

— Processes and capabilities of methods and tools for domain design and application design are described by ISO/IEC 26552.

— Processes and capabilities of methods and tools for domain realization and application realization are described by ISO/IEC 26553.

— Processes and capabilities of methods and tools for domain verification and validation and application verification and validation are described by ISO/IEC 26554.

— Processes and capabilities of methods and tools for technical management are described by ISO/IEC 26555.

— Processes and capabilities of methods and tools for organizational management are described by ISO/IEC 26555.

5.2 Product line reference model

Software and systems product line (SSPL) engineering and management shall consist of domain engineering and application engineering life cycles and organizational management and technical management process groups (Figure 1). They should be loosely coupled. The domain and application engineering life cycles need to be synchronized and applicable to different life cycle models as the life cycles are performed in a variety of organizational and technical environments to meet different quality criteria and to achieve different organizational objectives. Organizational and technical management process groups are necessary to help organizations to establish and improve capabilities for nurturing their product lines from conception to retirement and for establishing and managing relationships with customers, providers, and other key stakeholders.

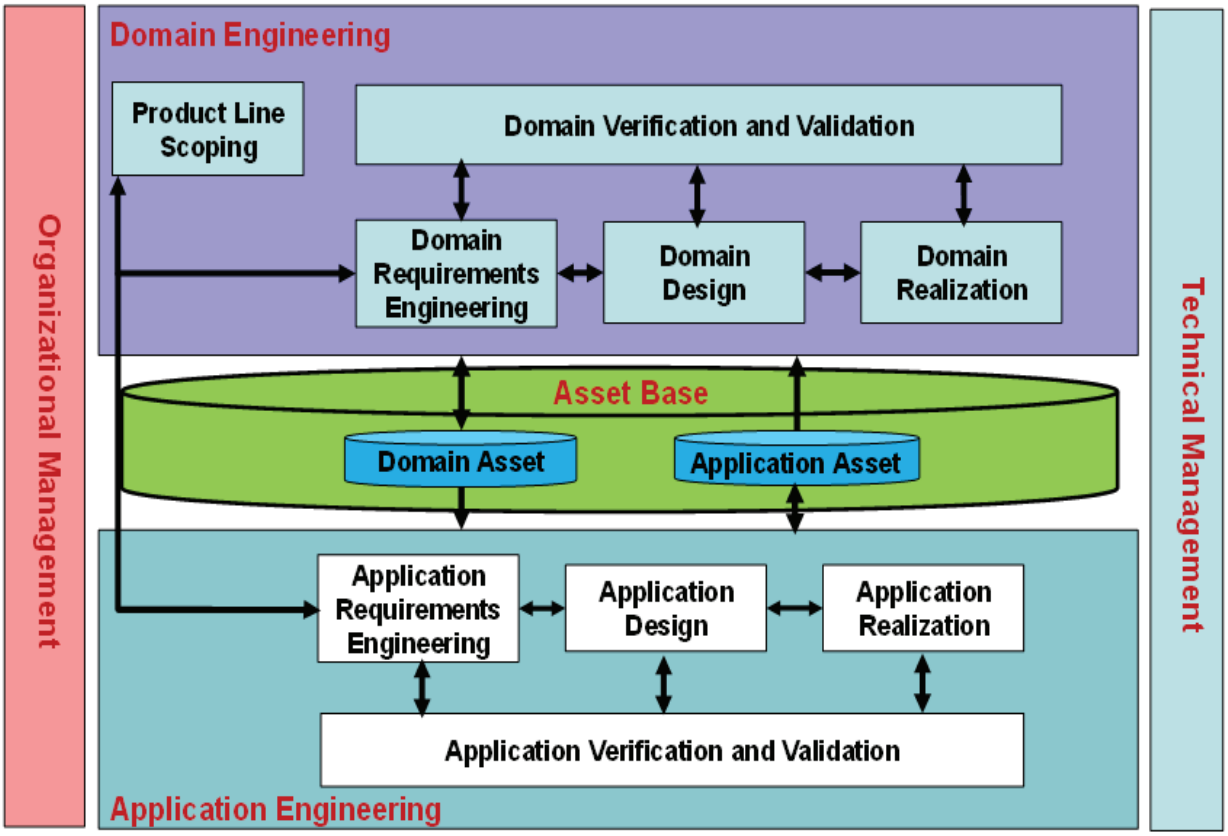

Figure 1 — Reference model for SSPL engineering and management

The two distinct engineering life cycles and the two process groups of Figure 1 are compatible with the process groups of ISO/IEC 12207 and ISO/IEC/IEEE 15288. The existence of two engineering life cycles is one of the major differences of product line engineering and management in comparison with single-system engineering and management. Therefore, the processes of the two life cycles make this difference explicit. Asset base stores both domain assets and application assets. It separates the two engineering life cycles and coordinates and synchronizes the processes within and across the life cycles. The technical management process group is in line with the project processes group, the software support processes group, and parts of the software reuse processes group of ISO/IEC 12207. The organizational management process group is in line with organizational project-enabling processes group and the agreement processes group of ISO/IEC 12207 and ISO/IEC/IEEE 15288.

NOTE 1 This product line reference model is not suitable for handling physical artifacts such as electronic boards, mechanical parts, or human operators. It is concerned with system and software level artifacts such as requirements documents, architectural data, validation plans, and behavioral models. It can be applied twice when there are software artifacts that belong to a larger system. It is used to deal with the system-level artifacts of the product line first and the software artifacts of the product line second. Moreover, the product line reference model is recursive within and across the different levels of software and systems engineering.

NOTE 2 The product line reference model and the International Standard as a whole do not prescribe the use of any particular software and systems engineering methodologies. Organizations can decide which methodologies to use for domain engineering and application engineering. Some organizations may use the same methodology for both domain and application engineering whereas others may choose to mix and match diverse (e.g. Domain Specific Modeling, Agile, and Lean) methodologies. The chosen methodologies should facilitate the strategic creation, reuse, and configuration of not only domain realization assets but also all other domain assets such as requirements, design, and test assets.

6 Two life cycles and two process groups for product line engineering and management

In accordance with the reference model for SSPL engineering and management (Figure 1), this International Standard focuses on the domain engineering and application engineering life cycles and the organizational management and technical management process groups. They are explained in this clause.

6.1 Domain engineering life cycle

Domain engineering identifies the key commonality and variability, sets up a common product line platform implementing the commonality and variability, and manages the platform throughout the life cycle of the product line. The underlying development methodology must thus enable the long-term development and maintenance of even large and complex platforms. Methods and tools for domain engineering facilitate the processes described in 6.1.1 to 6.1.5.

6.1.1 Product line scoping

Product line scoping defines the member products of a product line and the major (externally visible) common and variable features of the member products, analyzes the products from an economic point of view, and controls and schedules the development, production, and marketing of the product line and its products. Product management is primarily responsible for this process, consisting of three roles. This process has the following roles:

— *Product scoping* determines the product roadmap, that is (1) the targeted markets; (2) the product categories that the product line organization should develop, produce, market, and sell; (3) the common and variable features that the products should provide in order to reach the long and short term business objectives of the product line organization, and (4) the schedule for introducing products to markets.

— *Domain scoping* identifies the functional domains that are important to the envisioned product line and provide sufficient reuse potential to justify the creation of the product line. Domain scoping builds on the definitions of the product categories produced by product scoping.

— *Asset scoping* identifies potentially reusable assets; estimates the costs and benefits from each domain asset; and delivers these results as asset proposals for product line managers who decide which domain assets will be implemented. The existing assets that will be adapted into domain assets are documented in a list of assets.

NOTE Product line scoping is an organizational product line planning and management process. It could be placed in the organizational management process group (6.3). In this International Standard, it is presented as a domain engineering process because it strategically directs the execution of both domain and application engineering life cycles.

6.1.2 Domain requirements engineering

Domain requirements engineering uses the outcomes of product line scoping (e.g. the product roadmap, the specification of the product line's high-level features, and the documentation of existing assets) as the starting points for determining the common and variable requirements of the member products of the product line. It produces a requirements specification and the associated domain variability model and provides feedback, if necessary, to the product management about the changes required in the outcomes of product line scoping.

Common and variable requirements are elicited, analyzed, specified, verified and validated for the later domain engineering lifecycle processes, that is, domain design, realization, and verification and validation. This process has the following roles:

— *Domain requirements elicitation* determines and involves appropriate domain stakeholders, performs the requirements elicitation activities of single-system engineering, and captures the scope and

variations anticipated over the foreseeable lifetime of the product line explicitly through the use of appropriate techniques.

— *Domain requirements analysis* draws upon the defined product line scope and other relevant artifacts to perform the requirements analysis activities of single-system engineering and to find commonalities and identify variations. It may also clarify to application engineers and their clients, how a particular member product might achieve economic viability if it were able to use more common and fewer unique requirements. A domain variability model is also structured.

— *Domain requirements specification* documents the analysed product line requirements from which application requirements specification can select or adapt during application engineering.

— *Domain requirements verification and validation* ensure that all relevant product line requirements are analysed and specified completely, correctly, consistently, and unambiguously throughout the lifetime of the product line.

— *Domain requirements management* plans, defines, and manages the domain requirements engineering process and coordinates requirements engineering activities product line-wide together with product managers and other stakeholders responsible for application requirements management. It enacts a formal change management mechanism for proposing changes in the baselined domain requirements and for supporting the impact analysis of the proposed changes. Traceability links must be maintained between domain requirements and other assets (including requirements sources, application requirements, the domain architecture, and domain test assets) to facilitate impact analysis and appropriate changes in the assets impacted by changes in the requirements.

6.1.3 Domain design

Domain design develops a product line architecture that consists of the architectural structure and texture and enables realization of all member products. The domain architecture reflects not only the external variability but also the internal variability introduced by the chosen technical solution(s).

Domain design is broadly divided into four parts: commonality and variability analysis in domain design, domain architecture design, domain architecture evaluation, and domain architecture management.

— *Commonality and variability analysis in domain design* analyses and takes into account the commonality and variability in requirements to help design the domain architecture. The commonality and variability in requirements are elaborated by considering the domain architecture in terms of the structure and texture of software and systems.

— *Domain architecture design* captures the high-level design of a product line. It should satisfy the common requirements and support the key variable requirements to facilitate mass-customization of member products. Architectural texture, which consists of common rules for design, realization, coding, and integration testing, has to be defined and provided to all member products in order to establish a common approach for dealing with the domain architecture. The architectural texture includes the component framework that provides the allowed configurations of components with additional restrictions.

— *Domain architecture verification and validation* review the domain architecture and assures whether the domain architecture supports the functional and non-functional domain requirements. It involves checking the architectural texture, including the component framework, and assures that commonality and variability in domain requirements are covered by the domain architecture and that the new (mostly internal, market-invisible) variability introduced during domain design is feasible. It includes architectural evaluation for assessing design decisions from the viewpoint of quality attributes.

— *Domain architecture management* maintains and manages the domain architecture and the associated domain assets throughout the life cycle of a product line. The major tasks for managing the domain architecture can be divided into three groups: configuration management, change management, and traceability management. Configuration management is critical because the domain architecture is bound into numerous configurations during application engineering

engagements. These configurations must be verified and validated, as incorrect configurations can cause problems or even product failures later on. When configuration management is set up, it has to consider the variability model from the very beginning. Otherwise configuration management becomes too complex later on. In sum, all configuration management needs must be handled by a single configuration management process. This process is a support process in the technical management process group (6.4.4). Change management has to be carefully orchestrated because the domain architecture must have a long life span and be responsive to long-term evolutions of markets and technologies while remaining as immune as possible to changes in the market conditions of individual members of the product line and to progressions of component-level technologies. Traceability links between domain requirements assets and the domain architecture, between domain realization assets and the domain architecture, between domain verification and validation assets and the domain architecture, and between the domain architecture and the application architectures derived from it should be maintained at a comprehensible level to enable change management and configuration management.

6.1.4 Domain realization

Domain realization deals with the detailed design and implementation of common and variable domain assets. It involves the building and buying of components and supporting infrastructure. The planned variability must be realized with adequate implementation mechanisms and the core components and interfaces have to be verified and validated based on the domain architecture and texture. Domain realization deals with five major issues: interface realization; identification, evaluation, selection, and integration of commercial off-the-shelf (COTS) components; component realization; the verification and validation of domain realization assets; and domain realization management.

— *Identification, evaluation, selection, and integration of commercial off-the-shelf (COTS) components*. COTS components (including open source components) can dramatically speed up domain realization, lower the costs, and improve the quality of the platform as a whole. Appropriate components must be identified and evaluated. In the product line engineering context, especially important evaluation criteria are how well a COTS component meets the known variability requirements and how easy it is to add new variability when necessary. If suitable COTS components cannot be found and/or bought, new components must be realized. Decision management (6.4.4) in the technical management process group supports COTS component-related decision making.

— *Interface realization* includes the internal design and coding of the interfaces of common and variable components to ensure the interfaces expose appropriate information related to commonality and variability. The level of interface details for the provided components and required components should be reviewed.

— *Component realization*. Components are the units used to compose whole member products, as described by the domain architecture. Program coding is done based on the texture of the domain architecture and the design of the component and its interfaces. In addition, component realization includes assets such as configuration mechanisms that help application developers to select and bind variants and build member products with the domain components and interfaces.

— The *verification and validation of domain realization assets* detect failures in domain components and interfaces that affect all member products dependent on the domain assets. Failures should be prevented by ensuring the quality of domain realization assets. This role reviews whether the implemented components and interfaces conform to the domain architecture, whether they obey the architectural texture and the configuration mechanisms of the component framework, and whether the programming style is compliant with applicable industry practices or standards chosen for the product line. It should be noted that the domain verification and validation processes (discussed in 6.1.5) are partly responsible for this role. The role is discussed here to emphasize its critical nature for product line realization and the responsibility of the domain realization experts to verify and validate the assets they have realized.

— *Domain realization management* systematically controls the changes and maintains the integrity and traceability of the components and interfaces throughout the software product line life cycle.

Configuration management is especially complicated and important here, because multiple versions and variants of components are used.

The difference between domain realization and the realization of a single system is that the reusable components of domain realization are loosely coupled and configurable, and do not constitute an executable, testable artifact. In addition, domain realization includes configuration mechanisms to realize the variability.

6.1.5 Domain verification and validation

Domain verification and validation ensure the right domain assets have been modeled, specified, designed, built, verified, and tested in the right way as prescribed in previous domain lifecycle processes. They also generate domain verification and validation assets that can be reused in application verification and validation. Domain verification and validation is partial because there is typically no single executable member product comprehensively covering the domain to be tested. Moreover, many variants are typically missing because it may be feasible to create them only when one or more member products actually need them. Appropriate strategies are thus necessary to test the common domain assets and at least the most important variable domain assets during domain verification and validation and to prepare domain verification and validation assets for reuse in application verification and validation.

Since verification and validation activities inherent in domain requirements engineering, domain design and domain realization have already been addressed in 6.1.2 to 6.1.4, only the following issues are included in the following:

— *Domain test planning* documents the scope, strategy, resources, and schedule of domain verification and validation activities. A domain test plan should include detailed domain verification and validation requirements, the domain verification and validation strategy chosen to deal with product line variability, test activities, and test completion criteria. The verification and validation schedule draws upon the product roadmap determining the schedule when the product line applications have to be ready for market launch.

— *Domain test design* specifies domain test procedures, test cases, test data, and test environments considering variability involved in domain assets. It draws upon various domain assets obtained from other domain engineering processes to construct appropriate domain verification and validation assets. The tasks of selecting and constructing domain verification and validation assets are conducted throughout the domain engineering life cycle. Depending on the kind of variability involved with each domain asset, more or less testing can be done on the domain asset and less or more on related application assets. Domain test design determines what is tested where and constructs domain verification and validation assets accordingly.

— *Domain test execution* applies the constructed verification and validation assets to the test objects such as components, subsystems (e.g. groups of components), and the realized product line platform according to the domain verification and validation strategy. During and after the execution, verification and validation results are created, documenting the applied test cases and scenarios, the objects under test, and the expected and actual verification and validation results in a repeatable and verifiable way. If sample applications have been built during domain verification and validation, only the common domain verification and validation assets and the variable domain verification and validation assets relevant to the sample applications are executed. Concrete domain test cases are typically available only for those items. Variability of the product line that is not present in any of the sample applications cannot be verified and validated during domain engineering.

— *Domain test closure and report.* After verification and validation have been completed, the results are analysed to discover defects in domain assets and their root causes. Finally, a domain test summary report is created to document which domain assets have been tested, which verification and validation assets have been used, and which results have been achieved. The tests not covered by domain test execution are the responsibility of application verification and validation. Test cases performed in domain verification and validation typically have to be repeated in application verification and validation.

— *Domain test management* keeps track of domain verification and validation assets, such as defects and the change history of the assets.

6.2 Application engineering life cycle

Application engineering develops application assets and individual systems on top of a platform. Application engineering is effective and efficient in comparison to single-system engineering because, depending on the scope and maturity of the platform, much or even most of the product line engineering effort and complexity have been allocated to domain engineering, reducing application complexity and shortening application development times. Application engineering typically involves customers and thus needs to deal with evolving market needs.

6.2.1 Application requirements engineering

Application requirements engineering develops application-specific requirements reusing common and variable requirements defined during domain requirements engineering. It also has an important role in providing insights and feedback to domain requirements engineering in order to guide platform development especially in the early phases of the product line creation.

— *Application requirements elicitation* identifies stakeholders relevant to the member product, elicits initial requirements from the application stakeholders, uses the variation points and variants to communicate to the stakeholders, lets the stakeholders select the variants that best meet their needs, and binds the appropriate domain requirements of the product line based on the selections.

— *Application requirements analysis* first ensures that all initial requirements of the application stakeholders are understood and scrutinized for incorrectness, omissions, and inconsistencies through abstracting, modelling, prototyping, simulation, and other means. Stakeholder needs that cannot be fulfilled by the reuse of domain requirements, that is, the gap between domain and application requirements, are then analysed and negotiated.

— *Application requirements specification* documents the analyzed application requirements by adding the application-specific requirements to the specification of the selected, possibly adapted, and bound domain requirements. Application requirements should have well-structured documentation for the later use in subsequent application engineering lifecycle processes and for the incorporation into domain assets if necessary.

— *Application requirements verification and validation* confirm that the application-specific requirements are complete, correct, consistent, and unambiguous and that the bound variants are relevant to the specific product requirements. It should be noted that this role is partly the responsibility of the application verification and validation process. It is presented here to emphasize its critical nature for application requirements engineering and the responsibility of the application requirements engineers to verify and validate the application requirements artifacts in collaboration with product management and application verification and validation experts.

— *Application requirements management* plans, defines, manages, and coordinates the application requirements engineering process and baselines the application requirements. It provides application stakeholders with a formal mechanism for proposing new application requirements and changes in the baselined application requirements and for assessing the impacts of the proposed changes. Traceability links must be maintained between application requirements and other assets including application requirements sources, domain requirements, application architecture, and application verification and validation assets because changes in the application requirements can impact the related assets and changes in the other assets can impact the application requirements.

6.2.2 Application design

Application design derives an application architecture from the domain architecture in order to meet application requirements. Product specific adaptations shall be done to satisfy product specific requirements. The application architecture should adhere to the structure and texture of the domain architecture. This process has the following roles:

— *Binding variants of the domain architecture* is the responsibility of the application architect to establish a baseline for the application architecture. Variants for the variation points of the domain architecture must be bound according to the application variability model and the binding results from application requirements engineering. Otherwise, the application architecture may lack support for the features and quality properties application requirements engineers intended the member product to have.

— *Application specific architecture design*. The application architecture should be derived from the domain architecture to maximum possible extent. Otherwise, full benefits from the domain architecture cannot be reached. For example, the member product may be too different from the other member products and prove inappropriate for the product line. Yet, application architects typically have to design new application design assets. Appropriate domain design assets will be incorporated in the member product through the binding of the variation points with the selected variants of the domain architecture. The application architecture is a combination of the application design assets and the bound domain architecture. It is important to note that the new application design assets can be generalized later into domain design assets.

— *Application architecture verification and validation* ensure the application architecture has been composed correctly, complies with architectural rules, and meets the application requirements. They also check that the variability introduced during application design is feasible, appropriately documented in the application variability model, and can be bound correctly. The application architecture is also evaluated from the viewpoints of application-specific quality attributes. Domain verification and validation should have already ensured through the specification, implementation, and execution of domain verification and validation assets that the appropriate variability of the domain architecture can be bound correctly. However, application verification and validation need to design and implement new application verification and validation assets corresponding to the application design assets and execute them together with the domain verification and validation assets to assure the integrity and validity of the application architecture. It should be noted that application verification and validation are partly responsible for this role. The role is presented here to emphasize that application architects need to take responsibility for application architecture verification and validation and work in collaboration with application testers to ensure the application architecture meets expectations.

— *Application architecture management* manages and maintains the application architecture and the associated application assets throughout the life cycle of the member product. Feedback from application architects related to the domain architecture is also managed here. The major tasks for managing the application architecture can be divided into three groups: configuration management, change management, and traceability management. Configuration management is important because the member products may offer plenty of variability for mass-customization. Each member product can thus exist in numerous configurations that must be verified and validated. Change management supports and extends the application life cycle by dealing with changes in the market conditions of individual members of the product line and progressions of component-level technologies. Traceability links between application requirements assets and the application architecture, between application realization assets and the application architecture, between application verification and validation assets and the application architecture, and between the domain architecture and the application architecture should be maintained at a comprehensible level to enable change management, configuration management, and the provisioning of feedback to product line architects.

6.2.3 Application realization

Application realization implements member products by drawing upon the application requirements and architecture; reusing and configuring domain components and interfaces; identifying, selecting, and integrating appropriate commercial off-the-shelf components; and building new components and interfaces to enable product-specific functionality. It deals with the following roles:

— *Binding component-level variability*. Internal variation points of domain components are bound. Interfaces realized in domain realization should be reused without changes. Otherwise, the texture of the domain architecture is likely to be broken and the strategic reuse of domain assets is hampered.

— *Identification, evaluation, selection, and integration of COTS (including open source) components*. When domain components are inadequate or unavailable for application realization, COTS components are often a viable option. They can speed up application realization, lower the costs, and improve the quality of the member product**s**. Compared to application-specific components, COTS components may also be easier to generalize later on as parts of the platform, if necessary. Appropriate COTS components must be identified and evaluated. If suitable ones cannot be found and/or bought, new components must be realized.

— *Application specific interface realization* includes the internal design and coding of the interfaces of application-specific components. This should be conducted only when domain realization has not provided appropriate interfaces for reuse. The level of interface details for the provided components and required components should be reviewed carefully.

— *Application specific component realization* takes place precisely as in single-system engineering when there are no suitable domain assets and COTS components available to meet application requirements.

— The *verification and validation of application realization assets* review both the domain realization assets associated with the bound variants and application components and interfaces to see (1) whether the bound variants and implemented components and interfaces conform to the application architecture and obey the configuration mechanism(s) and architectural texture and (2) whether the programming style is compliant with applicable industry practices or standards chosen for the product line. It should be noted that application verification and validation are partly responsible for this role. The role is presented here to emphasize that application realization experts need to take responsibility for the verification and validation of application realization assets and work in collaboration with application testers.

— *Application realization management* defines, schedules, and coordinates the application realization process and systematically controls changes and maintains the integrity and traceability of application components and interfaces throughout the application life cycle. Configuration management is important because each member product evolves throughout its life cycle and retains variability that different stakeholders such as member product users can bind. Each member product can thus exist in multiple configurations during its life cycle. For maintenance, it is essential to know the versions of all domain components and interfaces used in the member product. Domain realization may provide member products with new domain asset versions that significantly refresh the member products (especially those having relatively long life cycles) from technological and/or other viewpoints. Traceability links must be maintained between application realization assets and other assets including the application architecture, the application variability model, domain realization assets, and application verification and validation assets.

6.2.4 Application verification and validation

Application verification and validation ensure the right member product and the right application assets have been modeled, specified, designed, built, verified, and validated right. They validate the final product and its architectural and realization assets with respect to the application requirements. They draw upon domain verification and validation assets and create application verification and validation assets to ensure sufficient quality of the member products.

Since verification and validation activities inherent in the other processes of the application engineering lifecycle have already been addressed in 6.2.1 through 6.2.3, this subclause only includes the following roles:

— *Application test planning* produces an application test plan, documenting the scope, test strategy, test completion criteria, resources, and the schedule of application verification and validation activities. For directly reused domain requirements, the respective portions of domain test plans can be reused and adapted in the application test plan as necessary. For adapted domain requirements and new application-specific requirements, test planning has to be done from the scratch and incorporated in the application test plan.

— *Application test design* specifies and constructs application test procedures, test cases, test data, and test environments by reusing and adapting domain verification and validation assets. Common parts of domain verification and validation assets can be reused as they are. For the verification and validation assets that contain variability, the variation points must be bound based on the application variability model and the inputs from application requirements engineering, application design, application realization, and domain verification and validation. For application-specific parts, new assets have to be constructed. The effort required for generating application verification and validation assets depends on the size and scope of the member product, as well as the extent to which domain assets have been reused during application engineering. If a member product derives most features from the product line platform and the respective domain verification and validation assets have been extensively established for the platform, application verification and validation takes relatively little effort.

— *Application test execution* applies the application verification and validation assets to various test objects such as components, subsystems, and the member product according to the application verification and validation strategy. In addition to verifying and validating the correctness and completeness of the member product, the conformance of the member product to the architectural texture has to be assessed. The binding of variability and the configuration realized in the member product have to be verified and validated. The expected and the actual verification and validation results are documented together with the executed application verification and validation assets.

— *Application test closure and reporting.* When the above-mentioned roles of application verification and validation have been completed, the results are analyzed to discover defects in application assets and their root causes. Finally, a test summary report is created. It documents which assets have been verified and validated, which verification and validation assets have been used, and what the results have been. A domain test summary report is reused in reporting.

— *Application test management* keeps track of defects, application verification and validation assets, and the change history of application verification and validation assets.

6.3 Organizational management process group

Organizational management processes are necessary for the orchestration of the product line organization. Introduction and institutionalization of the product line strategy in an organization requires ongoing preparation, planning, execution, and improvement efforts.

6.3.1 Organizational-level product line planning

Organizational-level product line planning pertains to strategic organizational-level planning. It involves various types of plans such as product line transition plans, sourcing plans, and domain asset investment plans. One of its most important responsibilities is to make the go/no-go decision regarding the adoption of the product line strategy by analyzing the benefits to be gained and the efforts and investments required. It should analyze business values expected from a product line such as cost reduction, productivity improvement, quality improvement, reduced business risks, shorter time-to-market, and increased market share. Based on the previous information, organizational business management should establish business value targets to achieve by introducing the product line approach. Thereafter it should check whether the business value targets have been measurably achieved. If they have not been achieved, the organization must take necessary corrective actions.

In some cases it is impossible to conduct this analysis solely in planning mode. Execution of plans and improvements based on the results of execution are needed as well. In such cases it is typically useful to conduct first a short domain engineering process cycle and develop the first product as quickly as possible to grasp market share and to test whether it makes sense to devise a product line. Then in a feedback loop the developed application assets (e.g. requirements, designs, and software components) are generalized, adapted to provide adequate variability, and brought into the platform, leading to another domain engineering process cycle. Such a strategy should be planned during organizational business management. Process management should determine how the domain and application engineering processes will be adapted to leverage the revised platform.

This process has the following roles:

— *Business opportunity analysis* helps stakeholders to decide (1) whether to initiate a product line approach and (2) whether to include a particular product as a member of an existing product line. This includes the cost/benefit estimation and business value analysis. For successful business opportunity analysis, establishing a collaborative atmosphere among representatives from top management, marketing, product management, engineering, and the customer relationship management and user groups is essential. Markets have to be analyzed to make the initial adoption decision and to guide the evolution of the product line and the introduction of new products into the product line on an ongoing basis. Business opportunity analysis also needs to determine the magnitude of investments required to establish and operate the domain asset base, the knowledge management infrastructure, and the appropriate product line engineering and management processes over the life cycle of the envisioned product line. Business opportunities exist when the revenues from markets and other sources can be expected to provide adequate funding for the required investments.

— *Customer relationship management* refers to the exchange of information and other resources between designated representatives of the product line organization (e.g. marketing, a product manager, domain experts, a user group coordinator) and one or more particular interest groups of the customer (e.g. legal, financial or technical entities, operations, training) who take delivery of the product produced by the product line organization, introduce the product in the customer organization, and support the product throughout its life cycle in the customer organization. Managing the customer relationship also requires that a product line organization coordinates customer requirements with those of the product line and realizes the changes required by the customers as necessary. Especially in business to business settings product line organizations and their customers should aim at establishing and maintaining mutually beneficial long-term relationships.

— *Developing a sourcing strategy* serves to establish an actionable plan for achieving specific product line goals and outcomes through contracting for products and services. Sourcing helps obtain new domain assets and incorporate new products in the product line. The strategy must include a plan to manage the providers of COTS components and other third party deliverables. For example, such key providers should be determined that meet the requirements posed by the expected variability, lifetime, and evolution of the product line. Contracts with those providers should be secured. If open source COTS components are deemed necessary, an organizational role with a clearly defined process and set of responsibilities should be created to manage the relationships with relevant open source communities. Finally, the development of the sourcing strategy involves deciding whether the users and other third parties shall be able to provide, sell, and distribute third party applications that can be installed and used in some or all member products of the product line. If the decision is positive, the sourcing strategy must include a plan for establishing appropriate organizational, procedural, and technical support for the third parties (e.g. application engineering tools and quality assurance, marketing, and online distribution mechanisms).

— *Organizational transition planning* establishes an organizational capability to populate and nurture a product line. It plans for the initiation of the product line and specifies schedules and resources such as the organization, people, and the budget needed to establish and manage the product line. It also defines target goals to be achieved by adopting a product line approach. All this information is documented in the transition plan that is the main deliverable of the organizational transition planning process.

— *Organizational operations planning* establishes an operations plan that outlines how particular organizational units produce domain assets, define the production plans, and use the domain assets and the production plans to establish products and how the production operations are measured and monitored. The operations plan is an especially important artifact whenever the competitiveness of the product line organization is significantly dependent on having lower production costs than the competitors have. Operations management (6.3.3) details and executes the operations plan.

— *Organizational product line evolution planning* defines how the domain and application engineering processes will be continuously evolved to leverage the effectiveness and efficiency of the product line.

— *Value management planning* provides integrated measurement schemes to improve the effectiveness of transition plans and operations plans. The target business values defined in the business opportunities and their goals that should be achieved are planned.

NOTE This paragraph provides an example of smartphone product lines to explain the *Developing a sourcing strategy role.* The ability of the manufacturers of smartphone product lines to provide their customers with large numbers of typically inexpensive but high quality third-party applications such as games that run on the member products of the smartphone product lines is critical for the success of the product lines. Markets tend to favor smartphones that offer the widest range of high quality third-party applications. Smartphones are the member products in this example. Third-party applications cannot typically be considered member products of the product lines of the smartphone manufacturers. They extend the features available in smartphones but the product management units of the manufacturers often have limited control over the features. The providers of third-party applications can establish their own product lines that leverage the smartphones as platforms. For example, the success of a smartphone product line is likely to attract more and more third-party game providers offering game product lines that support the smartphone product line, positively influencing the success of the smartphone product line. Third-party games are member products of the game product lines in this example.

6.3.2 Organizational product line-enabling management

Organizational product line-enabling management plans for the initiation and evolution of the product line and designs the organizational processes, the structure of authority and responsibilities, and the infrastructure needed to establish and manage the product line. Training is a core role for both the initial product line adoption and the long-term evolution of the product line. Product lines will typically necessitate (1) numerous technological and organizational changes during their life cycles and (2) a great deal of coordination within and across organizational, functional, and project boundaries because many organizational units and external partners are involved.

This process has the following roles:

— *Structuring the product line organization* involves identifying the organizational charter and boundaries; designing work roles and responsibilities concerned with domain and application engineering and their interactions, organizational management, and technical management; allocating and assigning resources; monitoring organizational effectiveness; improving organizational operations; establishing inter-organizational relationships; and managing organizational transitions.

— *Training and human resource management* helps to ensure that the organizational units responsible for creating, operating, and evolving domain assets and member products have properly trained and qualified personnel for each work role. Training activities must be coordinated with other activities involved in product line adoption and evolution.

— *Organizational quality management* assures that product line platform, member products, and both domain and application engineering processes meet organizational quality objectives.

6.3.3 Organizational product line management

Organizational-level product line planning initiates the organizational transitioning toward product line engineering. During the organizational transitioning, organizational product line management gradually takes ever larger role to institutionalize product line engineering in the organization. It is concerned with the systematic evolution of both the product line organization and the product line from the as-is state toward the desired state. It secures necessary funding and other resources; maintains the budget and the structure of authority and responsibilities for the product line; defines and maintains the infrastructure and processes enabling product line engineering and management; and ensures everyone in the product line organization know their responsibilities and the means available for carrying out these responsibilities. It monitors and controls the product line; defines and maintains the production plan and schedule for coordinating platform and asset creation in domain engineering and product development in application engineering based on the product line strategy, work effort estimates, the budget, and the ongoing analysis of customer needs and market environments; manages product configurations; monitors sourcing engagements and revises the sourcing strategy as necessary; and provides existing customers with support and other services necessary for deploying the products.

This process has the following roles:

— *Product line evolution management* periodically analyzes the status and changing trends of customer needs, key competitors, technology, and other market environments. Product lines should be evolved to cope with these changes through evolving product line platform and/or aligning member products.

— *Deployment and innovation management* facilitate the successful implementation of the product line approach through effective assessment, improvement, and activation of the product line plans. Pilot projects may be conducted to evaluate the capability and readiness of a product line organization before the product line approach is broadly deployed. The organization should verify and validate whether it can achieve its objectives through the product line strategy.

— *Operations management* typically designs, measures, and monitors how particular organizational units produce and evolve domain assets, define and evolve the production plans, and use the domain assets and the production plans to establish products. Operations management follows and executes the sourcing strategy to also define which assets and services will be sourced from third parties. The resulting operations designs are important domain assets. The key measures should help operations management and the product line organization as a whole to track the progress and deliverables of product line efforts and take corrective actions as necessary to achieve organizational business value targets.

— *Organizational risk management* differs from technical risk management as it manages risks within and across multiple organizational units, functions, and products. Many risks may hamper the organizational implementation of product line engineering and management. They need to be proactively managed, so they materialize as seldom as possible and can be dealt with effectively in case of materialization. For example, product line engineering and management requires a great deal of communication, coordination, and control across organizational, functional, and project boundaries. Failures in this role are costly.

— *Organization-level monitoring and control* measures the actual progresses and results accomplished by introducing and evolving the product line approach. The key measures for the product line organization should be focused on the defined business values and their measurable goals.

6.4 Technical management process group

Technical management is necessary for the creation and evolution of product line processes, domain assets, and products. It consists of process management, variability management, asset management, and support management.

6.4.1 Process management

People from different organizational units work together to engineer and manage product lines. These efforts require a great deal of cooperation. Therefore, product line processes should be designed to be cross-functional and inter-organizational. Designing such processes is challenging because application engineering and domain engineering produce very different types of deliverables, typically requiring different types of development methodologies. For example, member products need to meet market needs, the time to market often has to be short, and application life cycles are often short. Domain assets typically have long life cycles, are generic to meet the needs of diverse member products, and must meet stringent quality requirements.

The strategy described in 6.3.1 places great emphasis on application engineering and testing the first product in the markets in the beginning and increases the emphasis of domain engineering gradually when the first products have proven successful. It is often effective because (1) it reduces the need for two parallel methodologies in the beginning of product line creation by utilizing primarily single-system engineering and (2) transitions toward product line engineering as quickly as the developments in the markets and within the product line organization allow.

The following roles of process management are necessary to establish and manage a product line organization's capabilities for implementing appropriate product line processes.

— *Applying process enabling processes for product lines* helps to establish and manage the readiness of process infrastructure. It enables to secure process leadership and resources for definition, assessment, and improvement of product line processes.

— *Domain engineering process definition* establishes and maintains a set of organizational standard processes and tailoring guidelines for product line engineering and management. It defines the common processes that all participants of a product line should follow. It also establishes the processes and development methodologies that domain engineers should use throughout the domain engineering life cycle. Since different organizational units must collaborate, the organizational standard processes of a product line organization are defined at multiple levels and are cross-functional.

— *Application engineering process definition* establishes and maintains the process that fits with the development of member products. It serves to define the member product specific processes for developing member products under the product line context. It also includes the definitions for the application engineering processes, development methodologies, and tools that should be applied for all member products. Application engineering should use and/or tailor the common application engineering processes defined here. It is important to note that the domain engineering process definition may impose certain constraints that application engineering processes always need to follow.

— *Definition and application of relationship management processes and supporting information systems* for collaborating with external providers and open source communities deals with the implementation of the sourcing strategy (6.3.1). It institutionalizes relationship management processes, organizational and technical support processes, and organizational entities responsible for the processes to conduct engagements with the external stakeholders such as COTS component providers and open source communities.

— *Applying process monitoring and control for product lines* helps to monitor the performance of the domain/application engineering processes and to control the corrective actions for fixing the deviations between planned and actual performances. For monitoring the performance of processes, the measures and metrics should be selected to allow control over the processes according to the measurement results.

— *Applying process improvement for product lines* helps to manage organizational process assessment and improvement based on the measurement results. Organizational process assessment and improvement should be systematic to migrate from an as-is state toward a desired state of organizational processes.

6.4.2 Variability management

Product line variability defines how member products are differentiated from each other. Variability management requires explicit documentation of variability through domain and application variability modeling. Variability modeling, documentation, and evolution throughout the domain and application engineering life cycles are supported by the following variability management roles:

— *Variability modelling* supports the creation and maintenance of detailed variability models using variability related information from domain and application engineering. It supports variability modeling using consistent notations. There are two types of variability models: domain variability models and application variability models. Domain engineering typically provides most of the variability information necessary for structuring the domain variability model. The model is refined and managed throughout the domain engineering lifecycle. However, application engineering provides variability information as well because each member product may offer plenty of variability. This information is documented in an application variability model. This model is also refined throughout the application engineering lifecycle. The levels of detail of variability information differ depending on the process (e.g. application requirements engineering) where the information is produced.

— *Variability mechanism* is a mean for realizing the variability of product line. Because variability is introduced at the whole domain engineering lifecycle phases and binding occurs at the whole application engineering lifecycle phases, variability mechanisms that support the determined binding times, rules and constraints, and domain artifacts should be provided. Variability mechanism used differs from domain artifacts even at the same lifecycle phase. And it differs from realization (e.g.

different programming languages), installation, and deployment methods. This role supports that domain engineers choose the right variability mechanisms in accordance with the determined binding times, domain artifacts, selected deployment methods, and etc.

— *Variability binding* supports the consistent exploitation of variability during application engineering, facilitating proactive and correct reuse of domain assets. It maintains information that domain and application engineers, as well as automated tools, need to resolve variability appropriately; resolves variability established during domain and application engineering; and documents variability resolutions, so it is possible to see later on how variability has been resolved (e.g. which variants have been selected for a particular member product). Variability binding helps maintain member products as it documents precisely which variability has been bound and how in a particular member product.

— *Variability documentation* supports stakeholders to document variability models in detail with annotations. It helps stakeholders to understand and/or bind variability during product line engineering. Annotations may provide justifications for providing certain variability while omitting some other variability from the domain assets, supporting the reuse of domain assets across the member products of a product line. The annotations for the domain variability model should help make binding decisions. The annotations for the application variability model are expected to provide the rationale of the binding results and newly added variation points, variants, dependencies, and constraints for the specific member product.

— *Variability tracing* enables the establishment and maintenance of traceability links between elements of the variability models and the associated domain and application assets. An economic analysis of the costs and benefits from traceability links should be conducted to decide on the appropriate level of detail for traceability management. Having too much or too little traceability is costly.

— *Variability control and evolution* manages the different versions of variability models and associated traceability links stored in the asset base. It also deals with the change requests related to variability models, so the product line variability can be managed as a strategic organizational resource.

6.4.3 Asset management

Asset management is responsible for storing, mining, and managing domain assets and application assets resulting, respectively, from the domain engineering of the platform and the application engineering of individual products. Asset management in the product line context is more complex than in single-system engineering because product line engineering requires comprehensive management of the domain assets, the application assets, and the relationships among them. Asset management controls the platform and other assets, including the attributes for mining application assets (for possible generalization of application assets into domain assets), the annotations necessary to reuse domain assets (e.g. glues, processes, and their descriptions that are attached to domain assets and prescribe how to use the assets), change requests and feedback, traceability links, and the versions of domain and application assets after they have been baselined.

The asset base is a common repository for domain and application assets and for traceability links between the assets, between assets and platform releases, and between assets and member products. Domain assets together with the domain architecture and the configuration management plan comprise the platform of the product line and are stored into the asset base. Application assets are also stored in the asset base as some of them can be made generic and then reused as domain assets. The asset base supports the creation, storage, retrieval, update, versioning, and deletion of domain and application assets and traceability links. It is established and managed by asset management.

This process has the following roles.

— *Asset identification* identifies and evaluates domain asset candidates (e.g. features, models, specifications, and test cases) developed in, for example, application engineering or legacy application projects. It selects the best candidates to be made generic and reusable in order to populate the asset base. It is important to note that domain engineering only creates domain assets and stores all of them in the asset base.

— *Asset base implementation* constructs the asset base. Asset base should configure assets to make them easy to mine, retrieve, and manage. It should include annotations that provide necessary information which will be used for integrating and orchestrating assets for systematic reuse.

— *Asset verification and validation* ensure that the domain and application assets reflect the defined asset structure and can easily be mined, retrieved, and managed.

— *Asset evolution* manages and controls change requests, traceability, and versioning of assets after baselining.

6.4.4 Support management

Support management deals with the following roles that may support other processes and may provide the capabilities that can promote the success and quality of other processes. The roles of this process include technical quality management, configuration management, decision management, technical risk management, and tool management.

— *Technical quality management* ensures that work products and processes implemented in domain engineering and application engineering comply with the predefined provisions and plans. Functionalities and quality attributes intended to be achieved through the product line engineering are assessed through technical quality management schemes (e.g. quality assurance).

— *Configuration management* controls versions and releases of each application and domain asset. Configuration management is more complex in product line engineering and management than in single-system engineering and management because it should deal with the variability of product line platforms and member products.

— *Decision management* helps to choose the most beneficial option(s) among business or technical decision alternatives. The strengths and weaknesses implied by each alternative should be considered during decision making to choose the best option(s). This role serves all product line decision making including domain engineering, application engineering, organizational management, and technical management.

— *Technical risk management* addresses technical risk issues that could endanger the achievement of targeted business values and other product line objectives. Severe technical risks (e.g. lack of domain knowledge, uncertain or volatile domain requirements, lack of historical data for effort estimation) are likely to materialize without adequate technical risk management, so product line organizations must develop mitigation and/or contingency plans to cope with those risks.

— *Tool management* is needed to automate or semi-automate domain engineering and application engineering life cycle processes and the creation, configuration, tracing, verification, validation, maintenance, and evolution of domain and application assets.

7 Relationships within and between domain engineering and application engineering

This Clause explains the interrelations within and between the processes of domain engineering and application engineering defined in the reference model for product line engineering and management (Figure 1).

7.1 Interrelations between product line scoping and domain requirements engineering

The major results of product line scoping are product roadmaps and asset proposals. A product roadmap defines products that will be included within a product line and their major assets (e.g. high-level common and variable features that directly affect domain and application requirements engineering) with their quantified costs and estimated benefits. A product roadmap may define a schedule for delivering specific member products to customers or for bringing them to market. The schedule is the result of strategic reasoning and effort estimation performed jointly by the product line and/or product

managers. Each asset proposal includes a list of existing assets that could be used to derive domain requirements and other domain assets.

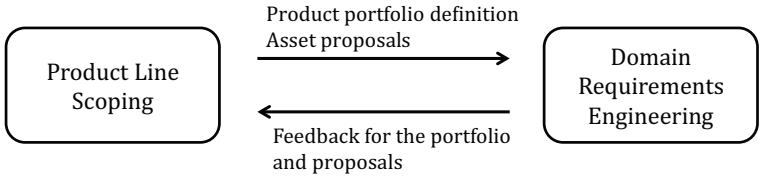

Figure 2 — Interrelations between product line scoping and domain requirements engineering

Features of the products of the product line are defined in domain requirements engineering. They deal mainly with external variability. They are communicated back to product line scoping as suggestions for additional and/or altered products or features, as well as feature refinements, based on the analysis of existing products, stakeholder needs, laws, constraints, and other requirement sources. The analysis may involve significant interaction with customers and providers. Corresponding processes should be established.

During the life cycle of the product line, product line scoping has to react to changes in customer needs, available technologies, competitors' offerings, and in other market conditions. These changes necessitate adaptations of the product roadmaps and asset proposals such as the introduction of new features or the elimination of outdated member products from the product portfolio.

7.2 Interrelations between domain requirements engineering and domain design

Domain requirements engineering has to follow the specification of the product line's high-level features provided by product line scoping to detail common and variable requirements that are adequate to steer domain design, realization, and verification and validation.

Domain requirements engineering provides to domain design all defined domain requirements and the definition of the product line variability in the variability model. Domain designers can then determine the technical solutions to be included in the domain architecture. The internal variability arising from domain design is added to the variability model produced by domain requirements engineering. The resulting variability model defines the variability of the domain architecture.

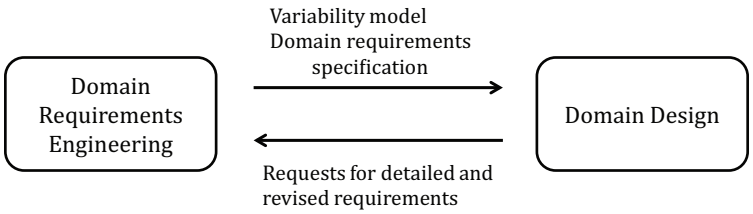

Figure 3 — Interrelations between domain requirements engineering and domain design

Domain design provides feedback to domain requirements engineering in terms of the needs for new, revised, or more detailed requirements.

7.3 Interrelations between domain design and domain realization

Domain design provides the domain architecture to domain realization. The domain architecture forms the basis for structuring all member products and the texture for building reusable components and interfaces.

Problem reports and other issues arising during domain realization are provided as feedback to domain design for improvement purposes.

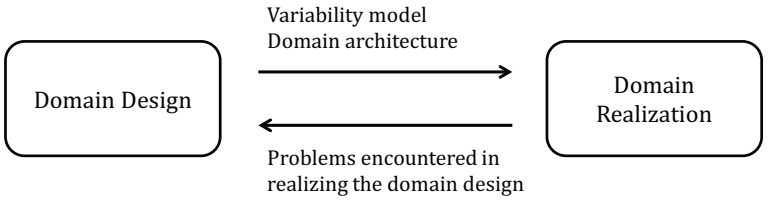

Figure 4 — Interrelations between domain design and domain realization

7.4 Interrelations between domain requirements engineering and domain verification and validation

Domain requirements engineering provides domain verification and validation with the domain requirements assets, specifying common and variable domain requirements, and the variability model. Domain verification and validation develop system tests and acceptance criteria for the acceptance tests and check whether the specified domain requirements are testable.

Platforms contain sets of loosely coupled components but no complete member products. Domain verification and validation can thus usually perform system tests only on subsystems that realize common requirements and are not affected by the variability of the product line. To find test cases without including variability, a configuration of variants is required. However, in cases where there are limited numbers of variation points with relatively small numbers of known variants, it is possible to test each variant in sequence. For example, for a variation point with three variants, three tests are planned. Huge numbers of test runs are needed to test all combinations of selected variants at different variation points, but this may be sensible and feasible in some cases.

Defects originating, for example, from incomplete or ambiguous domain requirements definitions and found during the domain system testing are reported to domain requirements engineering so that the defects in the domain requirements assets can be corrected.

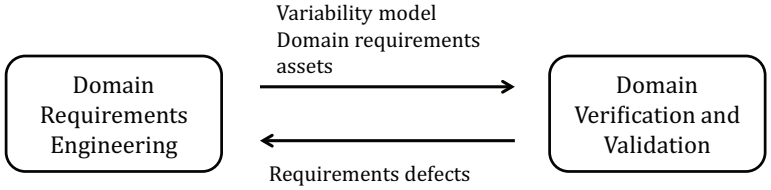

Figure 5 — Interrelations between domain requirements engineering and domain verification and validation

7.5 Interrelations between domain design and domain verification and validation

Domain design provides domain verification and validation with the domain architecture and the selection of reusable assets whose realizations are incorporated in the platform. Integration verification and validation assets are created for common component interactions and for those components that contain few variable interactions with realized components based on the domain architecture. Integration verification and validation assets should also be created at least for the most common interactions of variable components. Verification and validation assets must then provide the variability that matches the variability of the components and component interactions.

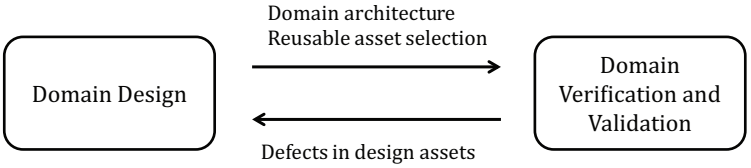

Figure 6 — Interrelations between domain design and domain verification and validation

Integration testing finds defects such as incomplete and ambiguous domain design assets, which prevent the definition of verification and validation assets. Defects originating in the domain design are reported to the domain design.

7.6 Interrelations between domain realization and domain verification and validation

Domain realization provides domain verification and validation with implemented reusable components and interfaces, as well as interface descriptions for unit testing. Integration testing is performed for the components that form a configuration specified in the domain architecture using the integration test specification formed in domain design. Domain tests can be performed only for the components that have been realized.

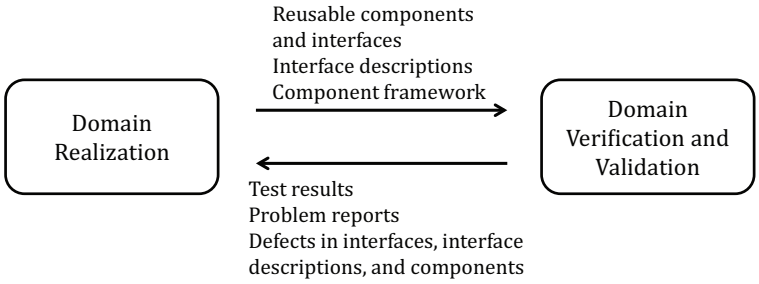

Figure 7 — Interrelations between domain realization and domain verification and validation

Domain verification and validation provide domain realization with the verification and validation results, which state whether the objects under verification and validation have passed or failed the test, the corresponding problem reports that describe in what manner the objects under verification and validation have failed, and defect descriptions.

7.7 Interrelations between product line scoping and application requirements engineering

Product line scoping specifies which member products should be derived in application requirements engineering by prescribing which member product should possess which of the common and variable features. Application requirements engineering refines the features prescribed by product line scoping.

Application requirements engineering provides product line scoping with feedback in terms of suggestions for additional or modified features, which result from the new insights gained.

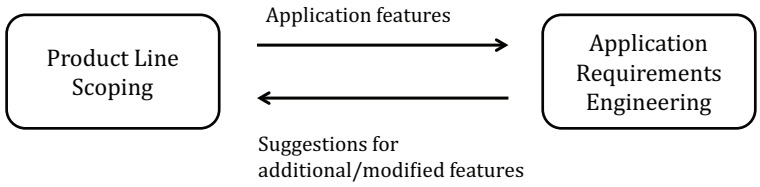

Figure 8 — Interrelations between product line scoping and application requirements engineering

7.8 Interrelations between domain requirements engineering and application requirements engineering

Domain requirements engineering provides application requirements engineering with the predefined common and variable requirements assets, as well as the variability model. The variability model supports the communication of the product line variability and the reuse of domain requirements assets.

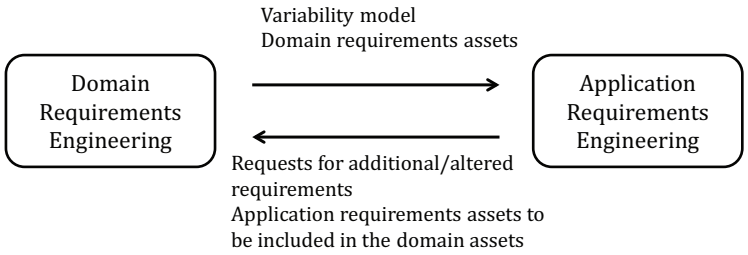

Figure 9 — Interrelations between domain requirements engineering and application requirements engineering

Application requirements engineering may offer additional and changed requirements and application requirements assets that domain requirements engineering may decide to incorporate in the domain assets. As a result, domain assets may change and the software product line may evolve over time. Product managers and other stakeholders responsible for the evolution of the product line will decide whether the input from application engineering affects the domain assets.

7.9 Interrelations between domain design and application design

Domain design supplies the domain architecture to application design that specializes the domain architecture for a single member product. Application design must obey the rules defined in the architectural texture.

Application design provides domain design with requests for additional and modified design assets. When the application architect finds that the architectural structure is insufficient for a particular member product, application design may provide domain design with design assets to be integrated into the platform. Such assets are newly developed parts of the application architecture that are of interest for the product line. Integrating them into the domain architecture contributes to product line evolution.

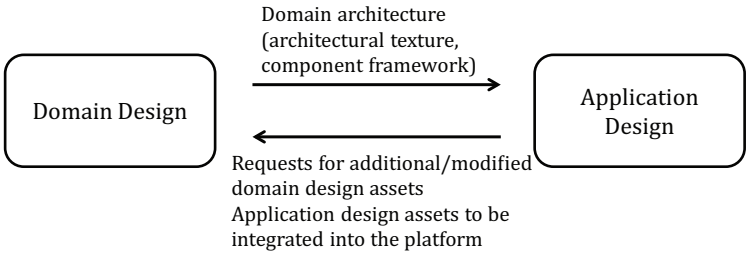

Figure 10 — Interrelations between domain design and application design

7.10 Interrelations between domain realization and application realization

Domain realization provides application realization with the reusable components and interfaces that have been designed, implemented, verified, and validated. It also provides application realization with configuration support to assemble the specific member products (e.g. by providing a configuration management tool).

Application realization provides domain realization with requests for additional and modified domain realization assets. This involves functionality or quality that should be provided by the domain assets but is not realized sufficiently well or at all. Application realization provides domain realization with application-specific components and interfaces that seem useful for the product line realization as a whole. Domain realization decides whether to incorporate these application assets into the product line.

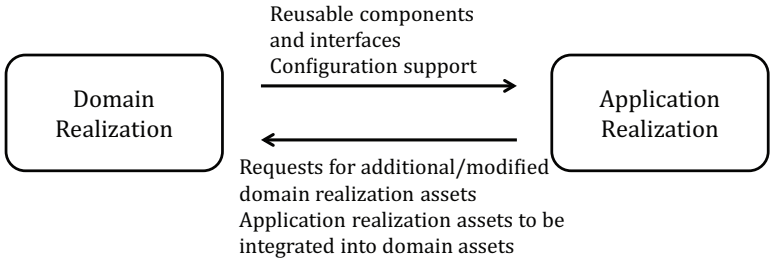

Figure 11 — Interrelations between domain realization and application realization

7.11 Interrelations between domain verification and validation and application verification and validation

Domain verification and validation provide application verification and validation with reusable verification and validation assets such as test cases. All domain verification and validation assets, including those already used in domain verification and validation, are delivered to application verification and validation. Like other domain assets, domain verification and validation assets may contain variability. Application verification and validation bind the variability to obtain verification and validation assets for the specific member product. Parts affected by the bound variants are retested according to the results of impact analysis even if they had already been tested during domain verification and validation.

Application verification and validation provide domain verification and validation with defects in domain verification and validation assets, as well as application verification and validation assets to be integrated into the domain verification and validation assets. Application verification and validation assets are developed, for instance, to test application-specific features. If application verification and validation assets are integrated into the domain assets, the related test, design, and realization assets have to be integrated as well.

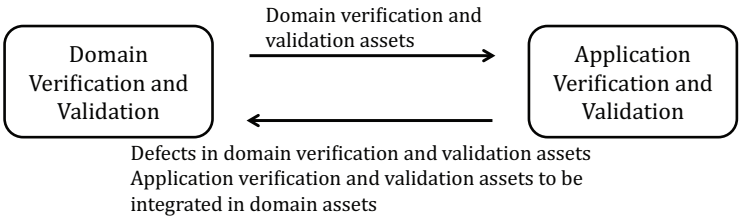

Figure 12 — Interrelations between domain verification and validation and application verification and validation

7.12 Interrelations between application requirements engineering and application design

Application requirements engineering provides application design with the application requirements specification. including the application variability model, which is derived from the binding of appropriate variability defined by the domain variability model and the incorporation of new variable features that differentiate the member product from other member products in the market and enable the mass-customization of the member product to market needs; application requirements assets some of which may be reused or adapted from domain requirements assets; and the requirements deltas. Requirements deltas are determined by analyzing and comparing the application requirements posed by the customer or the product manager with relevant domain requirements assets. Application design derives the application architecture based on the application requirements specification and the domain architecture provided by domain design.

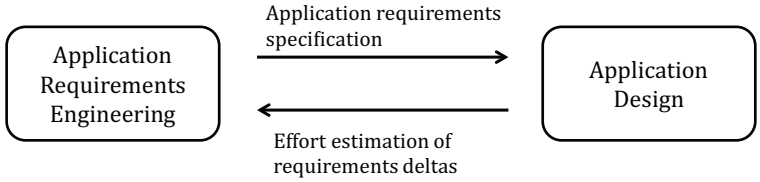

Figure 13 — Interrelations between application requirements engineering and application design

Application requirements engineering typically involves trade-off decisions with regard to the requirements posed by customers. The realization effort for the requirements depends on the degree of reuse that can be achieved. Requirements deltas such as performance requirements that are tighter than anticipated by the product line engineers may involve significant modifications of the domain architecture and the reusable components. Modifications affect the application engineering costs and it is necessary to decide whether to accept higher prices and/or longer schedules or to abstain from costly requirements.

7.13 Interrelations between application design and application realization

Application realization builds the member product based on the application architecture and domain realization assets. The application architecture determines the structure of the member product to be built, as well as the rules on how to build it, which are contained in the texture. The application architecture also determines the configuration of reused domain components and interfaces that are part of the member product, as well as their interrelations with application components and interfaces.

Application realization provides application design with design errors that emerge during realization and have to be solved by the architects. These include, for example, application components and interfaces that prove unusable and configurations that do not work properly.

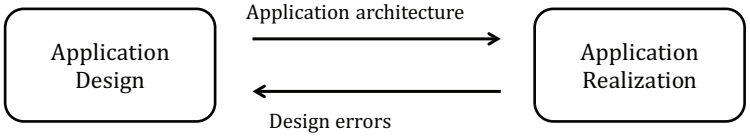

Figure 14 — Interrelations between application design and application realization

7.14 Interrelations between application requirements engineering and application verification and validation

Application system testing validates the member products against the application requirements specifications. System tests must ensure that the member products realize all the application requirements. When the variants for the member product have been selected and bound according to the variability bindings in the application variability model, application verification and validation will bind the respective variation points in the domain verification and validation assets and create new application verification and validation assets as necessary to validate application requirements.

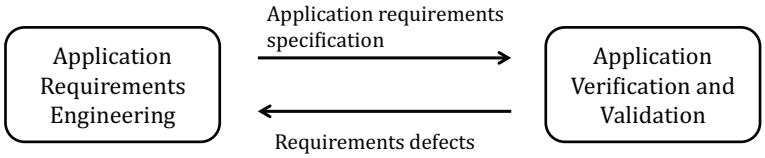

Figure 15 — Interrelations between application requirements engineering and application verification and validation

Application verification and validation report requirements defects such as ambiguous or incomplete requirements, as well as errors in the configuration of variants to application requirements engineering.

7.15 Interrelations between application design and application verification and validation

The application architecture created by application design drives application integration testing. The components composing the member product can be separated into domain components and application components, including modified domain components and new application-specific components. Even if the domain components had been tested, they should be retested by using appropriate domain verification and validation assets because the composition may have caused unforeseen defects and side effects.

During application integration testing, the application design is validated. Test engineers must ensure that the variability in the design has been bound properly and that the application design is testable. Whenever the test engineers cannot fully determine the integration test cases or the data required for executing the tests, an incompleteness or ambiguity is detected. Any defects in application design assets are reported to application design.

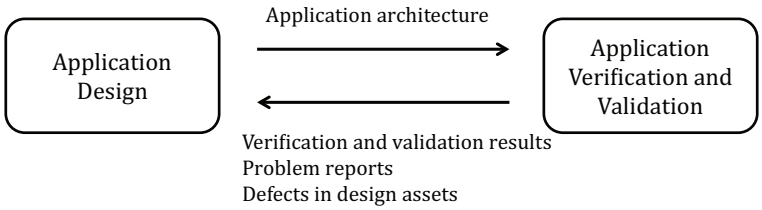

Figure 16 — Interrelations between application design and application verification and validation

7.16 Interrelations between application realization and application verification and validation

Application realization provides application verification and validation with an executable member product and associated application components and interfaces designed, implemented, and ready for unit testing.

Application verification and validation provide application realization with verification and validation results, problem reports, and defects in application interfaces and components.

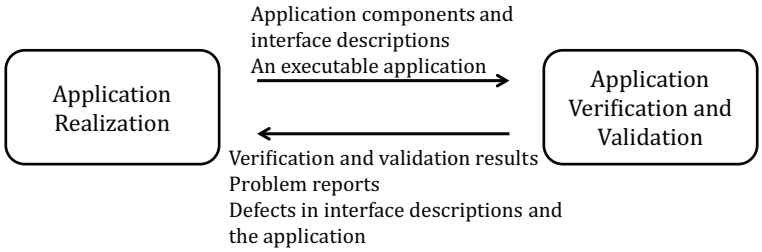

Figure 17 — Interrelations between application realization and application verification and validation

Annex A
(informative)

Further information on products

This International Standard aims at being applicable for both systems engineering and software engineering. However, it must be reminded that systems engineering produces products dealing with architectural design and does not encompass the implementation-related details, which are the responsibility of the respective technological disciplines such as mechanics, electronics, chemistry, and optics. Software engineering is one of these implementation disciplines, which creates computer programs. Due to the non-physical nature of software engineering, this discipline can reuse the same concepts for software product line engineering and management. Dealing with the other technological disciplines is beyond the scope of this International Standard.

An example of the levels and types of assets managed by the International Standard is given below:

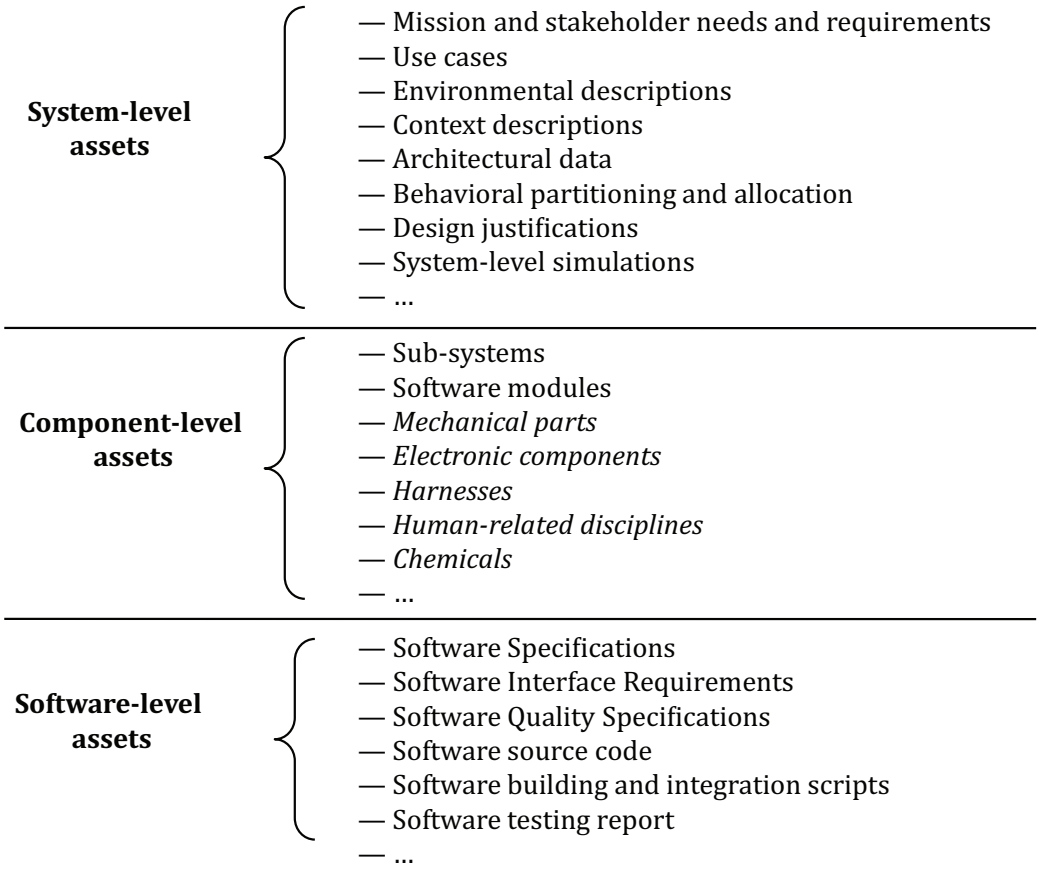

Figure 18 — Example of the levels and types of assets

The assets described in *italics* are outside the scope of this International Standard.

Bibliography

[1] ISO/IEC/IEEE 24765:2010, *Systems and software engineering — Vocabulary*

[2] ISO/IEC 15940:2006, *Information technology — Software Engineering Environment services*

[3] ISO/IEC 14102:2008, *Information technology — Guideline for the evaluation and selection of CASE tools*

[4] ISO/IEC 25000:2005, *Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE*

[5] ISO/IEC/TR 19759, *Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK)*

[6] Klaus Pohl, Günter Böckle, Frank J. van der Linden. Software Product Line Engineering: Foundations, Principles and Techniques. Springer 2005.

[7] Northrop Linda M., & Clements Paul C. A Framework for Software Product Line Practice, Version 5.0. Software Engineering Institute, Carnegie Mellon University, July 2007.

[8] Käkölä T., & Duenas J.C. Software Product Lines — Research Issues in Engineering and Management. Springer, 2006

[9] Bosch J.Design and Use of Software Architectures. Adopting and Evolving a Product-Line Approach. Addison-Wesley Professional, 2000

[10] Clements P.C., & Northrop L.M. Software Product Lines: Practices and Patterns. Addison-Wesley Professional, 2001

[11] van der Linden F.J., Schmid K., Rommes E. Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer, 2007

[12] ISO/IEC 12207:2008, *Systems and software engineering — Software life cycle processes*

[13] ISO/IEC/IEEE 15288:2015, *Systems and software engineering — System life cycle processes*

[14] ISO/IEC 26551, *Software and systems engineering — Tools and methods for product line requirements engineering*

ICS 35.080

Price based on 35 pages