INTERNATIONAL STANDARD

ISO/IEC 26550

First edition
2013-09-01

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 26550:2013(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

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

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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.

ISO/IEC 26550 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*.

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 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. The key characteristics of product line engineering are that there are domain and application engineering lifecycle processes and the explicit definition of product line variability. 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 and during application engineering they are exploited in accordance with the defined variability models.

The reference model for SSPL engineering and management can be used in subsequent standardization efforts to create high-level of abstraction standards (e.g. product management, scoping, requirements engineering, design, realization, verification and validation, organizational and technical management), medium-level of abstraction standards (e.g. configuration management, variability modeling, risk management, quality assurance, measurement, evaluation, asset repository), and detailed-level of abstraction standards (e.g. texture, configuration mechanism, asset mining) of 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 reference model fit together;
- define interrelationships between the components of the 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 standards (ISO/IEC 26551 through 26556). 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 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, structure from ISO/IEC 12207:2008, ISO/IEC 15288:2008, ISO/IEC 15940:2006 and ISO/IEC 14102:2008 has been used as a baseline.

2 Normative references

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

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

ISO/IEC 15288:2008, *Systems and software engineering — Systems life cycle processes*

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 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 processes that may be exploited in other lifecycle processes of the specific application engineering, and may be adapted as a domain asset based on a product management decision

NOTE 1 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 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 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 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
stores 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 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 Performing the binding is a task to apply the binding definition to generate new application assets 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 including the architectural structure and texture (e.g. common rules and constraints) that constrains all member products within a product line

NOTE 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. Syn: reference architecture, product line architecture.

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

Syn: core asset

NOTE 1 Domain assets may include domain features, domain models, domain requirements specification, domain architecture, domain components, domain test cases, domain process description, and etc.

NOTE 2 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 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 Domain assets are developed and managed in domain engineering processes and are reused in application engineering processes.

NOTE 2 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 Section 3 of IEEE 1517-2010 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
identifies and bounds 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 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
product belonging to the product line

Syn: application

3.16
product line
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

Syn: product family

3.17
product line architecture
synonym of domain architecture and reference architecture

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

NOTE Platforms have their own life cycles. ISO/IEC 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
defines the member products that will be produced within a product line and the major (externally visible) 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 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

3.22
reference architecture
synonym of domain architecture

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

NOTE 1 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 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 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 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
has two dimensions: variability dimension and asset dimension

NOTE 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
defines product line variability

NOTE It introduces variation points, types of variation for the variation points, variants offered by the variation points, and 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 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 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 is the dominant way of conceptualizing and developing software and systems products. This section first outlines some of the main challenges software and systems product companies face in using single-system engineering approaches. It identifies variability management as the most challenging area. Variability management is discussed in the second subsection. The section concludes by explaining major differences between single-system engineering and product line engineering and management. Understanding those differences is a key for successful organizational transitioning from single-system engineering toward product line engineering and management.

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

The excessive use of single-system engineering 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 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 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 sections 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, reuse of knowledge is important. However, product line engineering fundamentally differs from single-system engineering because the creation, management, and reuse of domain assets and the product line platform as a whole are of strategic importance. Product line engineering 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 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, shown in Section 4.3, are strongly related to variability management.

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

The identification and analysis of key differentiators between single-system engineering and product line engineering and management will help organizations to understand the product line reference model (Section 5.2) and to formulate a strategy for successful implementation of product line engineering. 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 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 distinguishes product line engineering from single-system engineering. 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