INTERNATIONAL STANDARD

ISO/IEC 26554

First edition
2018-11

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

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

Reference number
ISO/IEC 26554:2018(E)

COPYRIGHT PROTECTED DOCUMENT

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

Published in Switzerland

Contents

Page

Foreword ... **vi**
Introduction ... **vii**
1 Scope ... **1**
2 Normative references ... **1**
3 Terms and definitions ... **1**
4 Abbreviated terms ... **2**
5 Reference model for product line testing ... **2**
5.1 Overview ... 2
5.2 Product line test management ... 4
5.3 Domain testing ... 4
5.4 Asset management in testing ... 5
5.5 Variability management in testing ... 5
5.6 Application testing ... 5
6 Product line test management ... **7**
6.1 General ... 7
6.2 Product line test strategy ... 8
6.2.1 Principal constituents ... 8
6.2.2 Define product line test goals ... 8
6.2.3 Identify and analyse risks in product line test ... 9
6.2.4 Establish product line test strategies ... 9
6.3 Product line test process ... 9
6.3.1 Principal constituents ... 9
6.3.2 Select and tailor domain test process ... 10
6.3.3 Select and tailor application-specific test process ... 10
6.4 Product line test planning ... 11
6.4.1 Principal constituents ... 11
6.4.2 Develop organizational test plan ... 11
6.4.3 Gain consensus on organizational test plan ... 12
6.4.4 Document and share organizational test plan ... 12
6.5 Product line test monitoring and control ... 12
6.5.1 Principal constituents ... 12
6.5.2 Initiate monitoring and controlling product line test progress ... 13
6.5.3 Monitor product line test progress ... 13
6.5.4 Control product line test progress ... 14
6.5.5 Report product line test progress ... 14
7 Domain testing ... **14**
7.1 General ... 14
7.2 Domain test initiation and design ... 15
7.2.1 Principal constituents ... 15
7.2.2 Domain test initiation ... 16
7.2.3 Domain test design for unit testing ... 17
7.2.4 Domain test design for integration testing ... 18
7.2.5 Domain test design for system testing ... 19
7.3 Domain test environment set-up and maintenance ... 21
7.3.1 Principal constituents ... 21
7.3.2 Set up domain test environments ... 21
7.3.3 Enable interoperability with other domain engineering environments ... 22
7.3.4 Maintain domain test environments ... 22
7.4 Domain test execution ... 22
7.4.1 Principal constituents ... 22
7.4.2 Domain static testing ... 23
7.4.3 Domain dynamic test execution ... 24

7.5 Domain test reporting ... 25
7.5.1 Principal constituents ... 25
7.5.2 Analyse domain test results ... 26
7.5.3 Create/update domain test reports ... 26

8 Asset management in testing ... 26
8.1 General ... 26
8.2 Domain test artefacts as domain assets ... 26
8.2.1 Principal constituents ... 26
8.2.2 Identify domain test artefacts managed as domain assets ... 27
8.2.3 Structure configuration and annotation for domain test assets ... 27
8.3 Application test artefacts as application assets ... 28
8.3.1 Principal constituents ... 28
8.3.2 Identify application test artefacts managed as application assets ... 28
8.3.3 Structure configuration and annotation for application test assets ... 29

9 Variability management in testing ... 29
9.1 General ... 29
9.2 Variability mechanism category in testing ... 30
9.2.1 Principal constituents ... 30
9.2.2 Identify variability mechanisms in testing by category ... 31
9.2.3 Guide the use of variability mechanism category by PL test strategy ... 31
9.2.4 Guide the use of variability mechanism category by test levels ... 32
9.2.5 Trace the usage status of variability mechanism category in testing ... 32
9.2.6 Update variability mechanism category in testing ... 33
9.3 Variability in test artefacts ... 33
9.3.1 Principal constituents ... 33
9.3.2 Define variability type in test artefacts ... 34
9.3.3 Define variability representation in test artefacts ... 34
9.4 Traceability of variability in test ... 35
9.4.1 Principal constituents ... 35
9.4.2 Define explicit links between variability in test assets and variability model ... 35
9.4.3 Define explicit links between application test assets and application variability model ... 36

10 Application testing ... 36
10.1 General ... 36
10.2 Application test initiation and design ... 37
10.2.1 Principal constituents ... 37
10.2.2 Application test initiation ... 37
10.2.3 Application-specific test design for unit testing ... 38
10.2.4 Application test design for integration testing ... 40
10.2.5 Application test design for system testing ... 41
10.3 Application test environment set-up and maintenance ... 42
10.3.1 Principal constituents ... 42
10.3.2 Set up application test environments ... 43
10.3.3 Enable interoperability with other application engineering environments ... 43
10.3.4 Maintain application test environments ... 44
10.4 Application test execution ... 44
10.4.1 Principal constituents ... 44
10.4.2 Application static testing ... 45
10.4.3 Application dynamic test execution ... 46
10.5 Application test reporting ... 47
10.5.1 Principal constituents ... 47
10.5.2 Analyse application test results ... 48
10.5.3 Create/update application test reports ... 48

Annex A (informative) **Exemplar product line test strategy ... 49**

Annex B (informative) **Execution of SSPL testing ... 51**

Annex C (informative) **Mapping of ISO/IEC/IEEE 29119-2 and ISO/IEC/IEEE 15288** **52**
Bibliography **54**

Foreword

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

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

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

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

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

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

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

Introduction

The main purpose of this document is to deal with the capabilities of methods and tools of software and systems product line (SSPL) testing. This document defines how methods and tools can support the software and systems product line-specific testing processes.

Product line engineering sets up a common product line platform including identified key variability and develops individual systems on top of the platform. Variability realizes flexibility among member products, and it is closely related to the purpose of the product line reuse. In addition to the verification and validation of commonalities, domain testing generates reusable test artefacts, so as to minimize test efforts in application testing. However, variability continues throughout the product line life cycle, and its resolution phase is diverse. Thus, the complexity of product line testing becomes high. Testing in product line engineering differs from testing in the single system development in the following aspects:

— there are two testing life cycles, domain testing and application testing;

— test cases generated during domain testing can be incomplete due to unresolved variability;

— integration and system testing should be executed in the absence of non-functioning components because even complete test cases can interact with components or subsystems that include unresolved variability;

— application testing should be performed by reusing domain test assets and avoid retesting what has been tested during domain testing;

— test cases including variability are executable at different stages because the binding times of variabilities differ; and

— regression testing is performed in both domain testing and application testing. In application testing, when variability bindings are conducted, regression testing is performed as necessary.

This document addresses the product line-specific testing processes with the guidance of a set of tools' and methods' capabilities for supporting testing in software and systems product lines.

This document is intended to benefit the groups of people that acquire, supply, develop, operate and maintain tools and methods of testing for software and systems product lines. This document can be used in one or more of the following modes:

— by an organization intended to implement product lines – to understand, adopt and enact the processes, tools and methods of testing for product line. This also helps the organization evaluate and select relevant tools and methods based on business and user-related criteria;

— by a tool vendor who facilitate or leverage product line engineering practices – to provide a set of tool capabilities that should be embodied in a tool for supporting testing of a product line.

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

ISO/IEC 26550, ISO/IEC 26551, ISO/IEC 26555, ISO/IEC 26557, ISO/IEC 26558 and ISO/IEC 26559 are published. ISO/IEC 26552, ISO/IEC 26553, ISO/IEC 26556, ISO/IEC 26560, ISO/IEC 26561, ISO/IEC 26562 and ISO/IEC 26563 are planned International Standards. The following list provides an overview of the series:

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

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

— processes and capabilities of methods and tools for domain realization and application realization are provided in ISO/IEC 26553;

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

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

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

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

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

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

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

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

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

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

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

1 Scope

This document, within the methods and tools of testing for software and systems product lines:

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

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

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

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

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

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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 26555, *Software and systems engineering — Tools and methods for product line technical management*

3 Terms and definitions

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

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

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

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

3.1
absent variants
variants that are not determined or developed at the specific time

3.2
application testing
sub-process of application engineering where domain test artefacts are reused to uncover evidence of defects in the application

3.3
application test asset
application-specific test asset that has reuse potential

3.4
aspect
special consideration within product line engineering process groups and tasks to which one can associate specialized methods and tools

3.5
domain testing
domain engineering phase whose role is to test domain artefacts

Note 1 to entry: Testing of artefacts related to a product line (domain) rather than to an individual product.

3.6
domain test asset
domain test artefacts that will be reused in *application testing* (3.2)

Note 1 to entry: Domain test assets can be reused in *domain testing* (3.5), e.g., test plans, artefacts of regression testing. In this document, however, the platform role of domain asset, which can be reused for member products, is the central theme.

3.7
domain test requirements
specific elements of a domain artefact that should be covered by *domain testing* (3.5)

Note 1 to entry: Domain test requirements cover functional and non-functional commonality and variability, and they include both their normal and error conditions.

3.8
product line test strategy
scope of *domain testing* (3.5) and *application testing* (3.2)

Note 1 to entry: Many variants may not be implemented in domain engineering. To cope with the impacts of un-implemented variants to domain integration testing and system testing, the appropriate test strategy should be established.

3.9
texture
architectural texture
collection of common development rules and constraints for realizing the applications of a product line

3.10
variability binding
selection of a certain variant for a variation point

3.11
variability in test cases
variability included in domain test cases that will be bound during application testing in order to derive concrete test cases

4 Abbreviated terms

CRS	commonality and reuse strategy
SAS	sample application strategy
SSPL	software and systems product line
UML	unified modelling language

5 Reference model for product line testing

5.1 Overview

The key characteristics of product line testing are that there are two test processes: domain testing and application testing, with variability included in domain artefacts. The major difference between

SSPL testing and single product testing comes from variability in domain artefacts. Handling variability is challenging and the decisions on how to deal with it are the starting point of SSPL testing. From this view, there are two types of test assets, domain test assets (test cases, procedures and scenarios that are produced during domain engineering) and application test assets (test cases, procedures and scenarios that are produced during application engineering for a member product).

Domain test assets should have forms that can be efficiently reused in application testing; and they should address variations. In addition, testing for a member product of a product line should devise a method to reuse domain test cases and minimize regression testing for the parts that are already tested in domain testing or tested by another member product. It is unusual for all variants to be implemented during domain engineering. So, domain testing should consider tests for non-executable domain artefacts due to the undeveloped variants, i.e. absent variants. Coping with absent variants is challenging because they can complicate integration testing and system testing, making system testing in domain engineering impossible in some cases. However, because the quality of domain artefacts affects the quality of all member products in a product line, test execution of parts linked to absent variants should be carefully handled during domain testing.

Variability is bound during application engineering. The binding times of variability can be widely separate, so integration testing and system testing for a member product are much more complicated. If testing is executed whenever binding occurs, a significant amount of effort will be required for developing test stubs and drivers. Therefore, the trade-off between defect correction costs and test development effort should be considered. Application testing deals with testing for application-specific parts and with re-testing or regression testing for the domain parts already tested.

Product line testing requires specific organizational level managerial supports and technical management supports, product line test management and variability management in testing. Product line test management provides managerial supports for domain testing, application testing and asset management in testing from the product line organization level. Variability management in testing, on the other hand, manages variability in domain and application test artefacts, including variability models from the testing perspective.

The reference model specifies the structure of supporting processes and sub processes for product line testing. As shown in Figure 1, product line testing can be structured in five processes. Each process is divided into sub processes to address product line realization issues; and each sub process is described in terms of the following attributes:

— the title of the sub process;

— the purpose of the sub process;

— the outcomes of the sub process;

— the inputs to produce the outcomes;

— the tasks to achieve the outcomes; and

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

NOTE 1 Test methods in this document include technique, pattern and template as well as procedure.

NOTE 2 The ISO/IEC/IEEE 29119 series defines “test methods” as “procedure used to derive and/or select test cases”.

NOTE 3 When the process, sub process, outcomes and tasks are listed or described in a sentence they are italicized in order to increase their visibility.

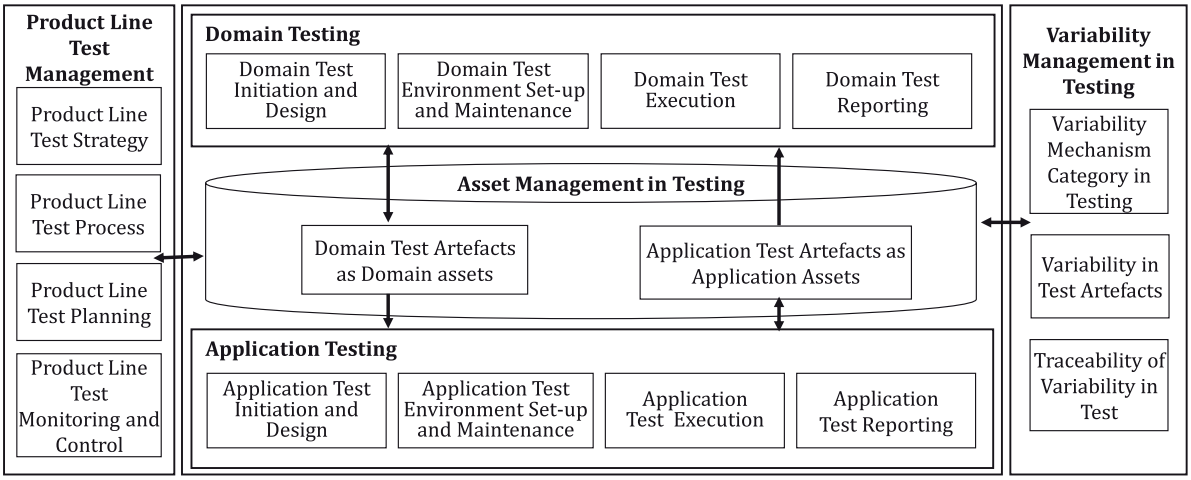

Figure 1 — Product line testing reference model

To reduce efforts and costs invested in product line testing and to manage the complexity of testing due to the variability in a product line, testing activities should be initiated as early as at the domain requirements analysis phase, in parallel with domain and application engineering. Defects in domain assets have influence on the whole product line members and dynamic testing is performed restrictedly because of variability, so static testing should be performed thoroughly. Product line testing thus covers both static and dynamic testing. Static testing for product lines includes reviews, inspections and prototyping for domain and application artefacts to fulfil verification and validation. In addition, testers responsible for domain and application testing should cooperate with domain requirements engineers, architects and developers in order to detect defects in the corresponding development phase as early as possible. To that end, these test organization should develop reasonable domain test plans and have systematic test case generation methods including tool supports at hand for each domain test levels or test types.

5.2 Product line test management

The product line test management shall be responsible for the following.

— *Product line test strategy* determines how domain and application testing will be carried out at the product line organization. The product line test strategy documents guidelines for governing product line testing operated in the organization.

— *Product line test process* is to define the process specific to a product line testing by selecting and tailoring a generic domain and application test process to solve domain and application-specific test problems.

— *Product line test planning* is to develop the product line test plan. It describes activities, schedules and resources for product line testing, and product line plans are revisited and refined as domain and application test plans for conducting the specific test types or test levels of domain and application testing.

— *Product line test monitoring and control* monitor and control the status of product line testing to ensure it is performed in line with the product line test plan. It serves to monitor, measure and control domain and application test progress. Domain and application test status are integrated into product line-wide management.

5.3 Domain testing

In domain testing, normally there is no single, executable application to be tested, since only the platform consisting of a set of loosely coupled components exists, and thus product line policies and processes reflecting this aspect are necessary. Domain testing covers the generation of domain test

artefacts and executions, and making them domain assets for reuse. Domain testing is responsible for validation and verification of domain requirements specification, domain architecture, detailed domain design artefacts and realization artefacts. However, system testing in domain engineering may be partially executed or may not be performed because there may be no complete application during domain engineering. Therefore, domain testing should have a proper strategy for these cases.

The domain testing shall serve to do the following and define the capabilities of tools and methods for supporting those services.

— *Domain test initiation and design* set up the readiness for performing domain testing, derive domain test cases and define domain test procedures.

— *Domain test environment set-up and maintenance* set up and maintain domain test environments including support tools as well as integrated environments for each lifecycle phase related to each domain test type (i.e. static test and dynamic test).

— *Domain test execution* performs dynamic and static testing for the domain on the established domain test environments. Domain test cases may include variability that cannot be executed simultaneously.

— *Domain test reporting* analyses the identified failures and locations where the failures have occurred and reports domain test incidents with how they will be managed.

5.4 Asset management in testing

The asset management in testing establishes and maintains domain test assets that have reuse potential for the right use in application engineering and application test assets for later reuse by the relevant member product or the other member products within a product line. The asset management in testing shall serve to do the following and to define the capabilities of tools and methods for supporting them.

— *Domain test artefacts as domain assets* manages domain test artefacts that will be reused by the product line members such as test cases for commonalities, those for variabilities and those including interactions between common and variable modules.

— *Application test artefacts as application assets* manages application test artefacts that will be referred in regression testing due to the application evolution or as inputs in other applications.

5.5 Variability management in testing

Variability is newly defined during domain testing for handling test requirements variation among member products. Domain test assets include variability because of the variability included in test targets (i.e. domain assets). Therefore methods and tools for product line testing should have the capability for variability representation and its binding at the right time. The variability management in testing shall serve to do the following and to define the capabilities of tools and methods for supporting them.

— *Variability mechanism category in testing* maintains a set of variability implementation mechanisms that can be used for expressing or realizing variability in domain test artefacts.

— *Variability in test artefacts* serves to define variability types and ways to express variability included in test artefacts such as test plans, test cases and test scenarios.

— *Traceability of variability in test* establishes and maintains trace links between variability in test artefacts and variability models in test defined separately. The trace links are required to support the reuse of test artefacts in application testing.

5.6 Application testing

Domain test assets should be correctly reused in application testing. The application testing validates the final member product against the application requirements by drawing upon test assets from domain test assets and by performing additional application-specific tests. This process shall guide to

minimize test efforts by avoiding redundant test execution and reusing domain assets. Furthermore, this process covers all test types and levels from static testing for application through acceptance testing. The application testing shall serve to do the following and to define the capabilities of tools and methods for supporting them.

— *Application test initiation and design* set up the readiness for performing application testing, derive application-specific test cases and define application-specific test procedures. Application test cases and test procedures are derived from domain test assets, so this process only deals with application-specific test cases and procedures.

— *Application test environment set-up and maintenance* set up and maintain application test environments based on the established domain test environment.

— *Application test execution* performs application dynamic and static testing on the established application test environments.

— *Application test reporting* analyses the identified failures and location where the failure has occurred. Application test reports include application-specific and domain test incidents as well as how they will be managed.

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

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

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

The following are the descriptions for each aspect concerning product line testing. The product line tests-relevant processes and tasks shall be identified on the basis of these aspects. The concerns specific to product line testing will enable an organization to understand the product line test-relevant processes, sub processes, tasks, methods and tools' capabilities.

— **Application engineering**. Application testing is a stage of application engineering where the domain test artefacts are reused. Application-specific testing is developed from scratch.

— **Binding**. Most application test artefacts are generated by variability binding according to application variability model, so testing processes, methods and tools for product lines should contain capabilities to support binding.

— **Collaboration**. For efficient and effective testing, preliminary artefacts for testing should be developed at each development phase. Hence, test managers/engineers and the corresponding participants of each development phase should collaborate with each other for producing test artefacts and ensuring testability of artefacts.

— **Configuration**. Since the test artefacts include product variations, configurations of test artefacts with the corresponding development artefacts should be consistently managed.

— **Domain asset**. Domain test artefacts such as test plans, test cases including test data and test suites that will be reused in the application testing are the major domain assets of product line testing.

— **Domain engineering**. Testing for product lines should be prepared and executed from the initial domain engineering phase combined with the characteristics of each domain engineering process.

— **Enabling technology support**. Proper technologies for producing domain test assets and for avoiding redundant testing in the member product testing should be supported.

— **Measurement and tracking**. The status of defects in domain assets should be carefully traced, because they have significant ripple effects on member products. Effectiveness and efficiency of domain testing and application testing should be measured for improvement.

— **Platform**. Testing should help ensure that the platform fulfils the specified domain requirements and its intended use.

— **Product management**. Testing evolves in accordance with the evolution and changes of product lines.

— **Reference architecture**. The reference architecture is the major input for designing domain integration testing.

— **Reusability**. Domain testing assets are developed as available artefacts as they are (or are partly) in a generic form so that a large part of the required testing in applications should not be developed from scratch.

— **Texture**. Domain testing and application testing should help ensure both of them conform to the corresponding guidelines and rules in the texture.

— **Traceability**. For each test artefact, a trace link is established to the corresponding references. For example, when system test cases are derived from use cases, each system test case is related to the corresponding use case by a trace link.

— **Cross functional validation and verification**. Detailed processes for validation and verification are defined in this document.

— **Variability**. Test artefacts contain variability, so that each member product selects variants to verify, validate and test its artefacts.

6 Product line test management

6.1 General

The major difference between SSPL testing and single product testing comes from variability. Thus, handling variability is very challenging; and the decisions on how to deal with it are the starting point of SSPL testing. This decision should be made from the organizational level by considering the efficiency and effectiveness of testing. From this decision, test strategy, test process and necessary test environments including organizational level support can be determined. A product line test management process includes the following four key sub processes.

— *Product line test strategy* provides technical guidelines for performing domain and application testing. Product line test strategy provides the scope of domain and application testing, which means that domain artefacts will be tested during domain testing and artefacts will be deferred to application testing (see Annex A for an exemplar product line test strategy).

— *Product line test process* is used to solve product line-specific test problems. Product line test process is defined by selecting and tailoring a generic domain and application test process.

— *Product line test planning* provides test plans for performing product line tests throughout the product line lifecycle phases. Domain and application test planning is based on domain and application engineering artefacts, i.e. requirements, architecture, detailed design and implementation artefacts, and the variability model of both domain and application engineering.

— *Product line test monitoring and control* help ensure that the overall product line test progresses are well aligned with test plans. They serve to monitor, measure and control domain and application test progresses. Domain and application test status are accounted by product line-wide test monitoring and control.

6.2 Product line test strategy

6.2.1 Principal constituents

6.2.1.1 Purpose

The purpose of this sub process is to establish the goals and strategies of domain testing and application testing. The strategies should consider the separation of domain testing and application testing and the existence of variability.

6.2.1.2 Inputs

The following inputs should be available to perform the product line test strategy process:

— organizational product line objectives; and

— test-related risks identified by organizational risk management.

6.2.1.3 Outcomes

The following outcome shall be available as a result of the successful implementation of the product line test strategy process.

— *Product line test strategy with its rationale* is established.

6.2.1.4 Tasks

The organization shall implement the following tasks with respect to the product line test strategy process.

— *Define product line test goals*: establish product line organization's test goals.

— *Identify and analyse risks in product line test*: review risks identified in product lines or additional risks, and identify those related to product line testing.

— *Establish product line test strategies*: define and maintain strategies on when and how to test variability in domain artefacts. This task includes reviews for the defined product line test strategies from effort, effectiveness and efficiency perspectives.

6.2.2 Define product line test goals

The goal of this task is to make product line test goals explicit.

The method should support defining product line test goals with the following capabilities:

— deriving test goals from organizational product line objectives and relevant factors; and

— formulating goals into the assessable forms.

A tool should support defining product line test goals by allowing the user to do the following:

— use documentation template for test goals; and

— share test goal with related test participants.

6.2.3 Identify and analyse risks in product line test

The goal of this task is to identify risks related to product line testing.

The method should support identifying and analysing risks in product line test with the following capabilities:

— identifying risks to be treated by product line testing from risks identified in organizational risk management; and

— identifying additional risks related to variability treatment in product line testing.

A tool should support identifying and analysing risks in product line test by allowing the user to do the following:

— perform a risk assessment and its mitigation plan for identified product line test risks; and

— record risks related to product line tests.

6.2.4 Establish product line test strategies

The goal of this task is to define test strategies to proceed domain testing that should deal with variability. Product line test strategy shall devise appropriate risk treatment approaches.

Domain test strategies should include the scope of regression testing, test selection criteria for regression testing and techniques to be used for regression testing.

The method should support establishing product line test strategies with the following capabilities:

— devising appropriate risk treatment approaches;

— defining product line test strategy alternatives including the scope of domain and application testing;

— defining test case selection criteria for domain and application testing in accordance with the product line evolution; and

— defining additional metrics used for evaluating strategies from product line engineering viewpoints.

A tool should support establishing domain test strategies by allowing the user to do the following:

— use test strategy templates;

— share product line test strategy among relevant test engineers;

— notify the update of test strategy;

— access the history data to determine test case selection criteria for domain and application testing; and

— perform data collection and integration for the defined metrics.

6.3 Product line test process

6.3.1 Principal constituents

6.3.1.1 Purpose

The purpose of this sub process is to define test processes that will be implemented by the product line test organization.

The process mapping results with ISO/IEC/IEEE 29119-2 and ISO/IEC/IEEE 15288 in Annex C can be referred to define test processes.

6.3.1.2 Inputs

The following inputs should be available to perform the product line test process:

— organizational standard processes for domain/application testing; and

— tailoring guidelines for processes.

6.3.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the product line test process.

— *Selected and tailored domain test processes*.

— *Selected and tailored application test processes*.

6.3.1.4 Tasks

The organization shall implement the following tasks with respect to the product line test process.

— *Select and tailor domain test process*: find adequate domain test processes to achieve test goals. Domain test processes are selected and tailored based on test strategies established.

— *Select and tailor application-specific test processes*: find adequate application test processes in accordance with different member product's needs.

6.3.2 Select and tailor domain test process

The goal of this task is to select and tailor organizational domain test process to solve problems that domain testing encounters.

The method should support selecting and tailoring domain test processes with the following capability:

— providing ways to support the right selection of the organizational standards of domain test process.

A tool should support selecting and tailoring domain test processes by allowing the user to do the following:

— access domain engineering process assets related to domain testing;

— access tailoring guidelines defined in domain engineering process definition; and

— utilize electronic domain test process documentation.

6.3.3 Select and tailor application-specific test process

The goal of this task is to define a application-specific test process to solve problems that application testing specifically encounters.

The method should support selecting and tailoring application-specific test processes with the following capability:

— providing ways to support the selecting and tailoring of application-specific test processes while reusing selected and tailored domain test processes.

A tool should support selecting and tailoring application-specific test processes by allowing the user to do the following:

— access selected and tailored domain test processes; and

— access application engineering process assets related to application testing.

6.4 Product line test planning

6.4.1 Principal constituents

6.4.1.1 Purpose

The purpose of this sub process is to establish a test capability to perform the product line test from the previous lifecycle phases. Testability of domain/application requirements, domain/application architecture, detailed domain/application design and domain/application implementation are reviewed and proper plans to proceed to testing should be established in accordance with the review results.

Plans for testing are harmonized with the plans defined in the organizational management of ISO/IEC 26556. Detailed activities and tasks for this process are harmonized with ISO/IEC/IEEE 29119-2.

6.4.1.2 Inputs

The following inputs should be available to perform the product line test planning process:

— domain/application requirements;

— domain/application architecture;

— detailed domain/application design; and

— domain/application implementation.

6.4.1.3 Outcomes

The following outcome shall be available as a result of the successful implementation of the product line test planning process.

— *Organizational test plans* are defined and shared.

6.4.1.4 Tasks

The organization shall implement the following tasks with respect to the product line test planning process.

— *Develop organizational test plan*: make test plans to carry out product line test and maintain its progress.

— *Gain consensus on organizational test plan*: get consensus for the test plan from relevant stakeholders including domain and application test participants.

— *Document and share organizational test plan*: record and share a test plan with relevant stakeholders.

6.4.2 Develop organizational test plan

The goal of this task is to define plans including a test plan, a risk mitigation plan in test and a test operation plan.

The method should support developing an organizational test plan with the following capabilities:

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

— tracking the progress based on plans.

A tool should support developing organizational test plan by allowing the user to do the following:

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

— share plans with the relevant participants; and

— perform progress tracking against the plans.

6.4.3 Gain consensus on organizational test plan

The goal of this task is to consult and agree with the stakeholders on the defined test plan.

The method should support gaining consensus on the organizational test plan with the following capability:

— providing opinion collection approaches efficiently applied at the distributed or large product line organization.

A tool should support gaining consensus on organizational test plan by allowing the user to do the following:

— distribute organizational test plans to relevant stakeholders; and

— collect suggestions related to organizational test plan.

6.4.4 Document and share organizational test plan

The goal of this task is to distribute the defined organizational product line test plan to stakeholders.

The method should support documenting and sharing organizational test plan with the following capabilities:

— providing documentation templates; and

— providing feedback mechanisms efficiently applied at the distributed and large product line organization.

A tool should support documenting and sharing the organizational test plan by allowing the user to do the following:

— recode the organizational test plan; and

— share the organizational test plan with relevant stakeholders including domain and application test participants.

6.5 Product line test monitoring and control

6.5.1 Principal constituents

6.5.1.1 Purpose

The purpose of this sub process is to monitor and control product line test progress.

6.5.1.2 Inputs

The following inputs should be available to perform the product line test monitoring and control process:

— domain test progress; and

— application test progress.

6.5.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the product line test monitoring and control process.

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

— *Corrective action list for test* is established.

6.5.1.4 Tasks

The organization shall implement the following tasks with respect to the product line test monitoring and control process.

— *Initiate monitoring and controlling test progress*: identify suitable measures and suitable means for monitoring and controlling product line test progress against the organizational test plan.

— *Monitor product line test progress*: measure the actual domain and application test progresses in line with the organizational test plan. Actual domain and application test progresses should be monitored separately and integrated into the whole progress.

— *Control product line test progress*: make corrective actions in accordance with control directives.

— *Report product line test progress*: share the test progress with the relevant stakeholders.

6.5.2 Initiate monitoring and controlling product line test progress

The goal of this task is to define the measures and approach needed to monitor and control the progress of the product line testing.

The method should support initiating monitoring and controlling product line test progress with the following capabilities:

— defining measures for monitoring and controlling test progress for domain testing;

— providing measures for monitoring and controlling test progress for large amount of variability; and

— providing treatment approaches for integrating test progresses for domain and application testing.

A tool should support initiating monitoring and controlling product line test progress by allowing the user to do the following:

— utilize (semi-)automated measurement environment.

6.5.3 Monitor product line test progress

The goal of this task is to check the progress of domain and application testing, based on assigned roles and responsibilities.

The method should support monitoring product line test progress with the following capability:

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

A tool should support monitoring product line test progress by allowing the user to do the following:

— utilize (semi-)automated measurement environment.

6.5.4 Control product line test progress

The goal of this task is to actualize corrective actions in tests to support the defined business value achievement.

The method should support controlling product line test progress with the following capabilities:

— decomposing corrective actions transferred from the higher level of management to assign roles and responsibilities to the relevant participants of test; and

— providing a regression test strategy for actualizing corrective actions in test.

A tool should support controlling product line test progress by allowing the user to do the following:

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

— interoperate with the other test types and levels in domain/application testing for regression testing.

6.5.5 Report product line test progress

The goal of this task is to generate and share reports for product line test progress.

The method should support reporting product line test progress with the following capabilities:

— defining the components of report for product line test progress; and

— providing documentation templates for reporting.

A tool should support reporting product line test progress by allowing the user to do the following:

— generate a test report; and

— share product line test progress with relevant product line test stakeholders.

7 Domain testing

7.1 General

Domain testing produces test assets that will be reused by member products within a product line. Domain testing includes testing for common parts related to variable artefacts that may or may not be realized during domain engineering. Meanwhile, application testing has to achieve an efficient reuse of domain test assets while it tests application-specific parts. Domain test assets are major inputs of application testing. Test assets such as test plans, test cases and test scenarios are produced in the relevant phase. To realize such test assets, test engineers initiate system testing in the domain or application requirements engineering phase, integration testing in the domain or application architecture design and unit testing in the domain or application realization phase respectively. Test assets can be produced and executed either in domain testing or in application testing. Domain testing process includes the following four key sub processes.

— *Domain test initiation and design* generate domain test cases and test procedures.

— *Domain test environment set-up and maintenance* set up and maintain domain test environments. Domain test environments shall be composed in detail based on test environments described in the test plan defined as a result of Product line test planning process.

— *Domain test execution* performs dynamic and domain static testing based on the test procedures generated during domain test design and implementation.

— *Domain test reporting* describes the identified failures and its locations.

7.2 Domain test initiation and design

7.2.1 Principal constituents

7.2.1.1 Purpose

The purpose of this sub process is to derive domain test cases and test procedures for domain test execution. This process defines domain test plans for testing domain artefacts and prepares domain test inputs such as test cases, test scenarios and test data. Because there are incomplete parts in domain test inputs due to variability, this sub process should handle variability and variability-relevant common parts.

Domain test cases can be created either directly from domain artefacts or through domain test models derived from domain artefacts. A domain test model is a test model that preserves variability while an application test model only preserves the values of variability. A domain test case may or may not include variability.

7.2.1.2 Inputs

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

— organizational test plans;

— domain test references (e.g. domain requirements specification and domain architecture specification); and

— domain variability models.

7.2.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the domain test initiation and design process.

— *Domain test requirements and conditions* are defined.

— *Domain test cases* are generated.

— *Domain test procedures* are defined.

7.2.1.4 Activities

The organization shall implement the following activities with respect to the domain test initiation and design process:

— domain test initiation;

— domain test design for unit testing;

— domain test design for integration testing; and

— domain test design for system testing.

7.2.2 Domain test initiation

7.2.2.1 Tasks

The organization shall implement the following tasks with respect to the domain test initiation activity.

— *Initiate domain testing*: set up the readiness for proceeding domain testing process.

— *Define domain test requirements and conditions*: define test requirements to be addressed in domain testing and test conditions to be set.

7.2.2.2 Initiate domain testing

The goal of this task is to develop domain test plans that include domain-specific objectives, the scope of the domain test, responsibilities, domain test strategies, methods, tools to be applied, schedules for domain tests and test completion criteria.

The method should support initiating domain testing with the following capabilities:

— defining entry criteria to be assured to proceed to domain/application test;

— defining entry criteria of domain test to proceed to application test;

— defining guidelines to evaluate the readiness;

— selecting artefacts to be evaluated to confirm the readiness; and

— defining documentation components and guide for documentation.

A tool should support initiating domain testing by allowing the user to do the following:

— access the artefacts to be evaluated; and

— share defined entry criteria with the relevant participants.

7.2.2.3 Define domain test requirements and conditions

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

The method should support defining domain test requirements and conditions with the following capabilities:

— analysing the characteristics of domain testing in accordance with the nature of systems and software domain;

— defining the form of test requirements to be used for deriving domain test cases, domain test data and test scenarios; and

— defining test architecture and test coverage in accordance with the defined test strategy.

A tool should support defining domain test requirements and conditions by allowing the user to do the following:

— access inputs for test design such as domain requirements specifications, domain architecture, detailed domain design specifications and codes;

— utilize modelling notation for test requirements; and

— utilize modelling notation for test architecture.

7.2.3 Domain test design for unit testing

7.2.3.1 General

This activity addresses the domain test design for unit testing.

7.2.3.2 Tasks

The organization shall implement the following tasks with respect to the domain test design for unit testing activity.

— *Create domain test cases for unit testing*: design and generate test cases including test data that will be used for unit testing for domain and reused for unit testing for member products.

— *Derive domain test procedures for unit testing*: define the orders of test cases executed together. This task includes an ordered set of test cases to be re-executed due to changes.

— *Develop auxiliary test implementation for absent variant-related unit testing*: develop mocks or stubs necessary to execute test procedures related to variability.

7.2.3.3 Create domain test cases for unit testing

The goal of this task is to design test inputs such as domain test cases and test data for unit testing and to be reused for application testing.

The method should support creating domain test cases for unit testing with the following capabilities:

— addressing variability in units for test case generation;

— providing appropriate variability mechanisms to be used in unit test cases;

— supporting relation management between domain test inputs and relevant domain units for test evolution in accordance with changes on variability in units; and

— providing ways for test selection for regression testing of domain units.

A tool should support creating domain test cases for unit testing by allowing the user to do the following:

— use the unit test cases repository;

— utilize notation for variability in domain unit test cases (including test data);

— document domain unit test cases (including test data); and

— select test cases for domain unit testing.

7.2.3.4 Derive domain test procedures for unit testing

The goal of this task is to collect unit test clues and then to define types of required domain unit testing.

The method should support deriving domain test procedures for unit testing with the following capabilities:

— providing variability treatment approaches for a variability-related set of test cases; and

— providing variability mechanisms to be used in test procedures.

A tool should support deriving domain test procedures for unit testing by allowing the user to do the following:

— utilize notation (description) for variability in test procedures.

7.2.3.5 Develop auxiliary test implementation for absent variants-related unit testing

The goal of this task is to select domain- or application-specific test inputs such as test cases, test data and test scenarios to be re-executed.

The method should support developing auxiliary test implementation for absent variants-related unit testing with the following capabilities:

— providing proper treatment for mocks or stubs for executing variability-relevant unit test cases and test procedures; and

— providing appropriate approaches for reusing mocks or stubs for unit testing.

A tool should support developing auxiliary test implementation for absent variants-related unit testing by allowing the user to do the following:

— utilize environment for implementing mocks or stubs for variability; and

— reuse mocks or stubs.

7.2.4 Domain test design for integration testing

7.2.4.1 General

This activity addresses the domain test design for integration testing.

7.2.4.2 Tasks

The organization shall implement the following tasks with respect to the domain test design for integration testing activity.

— *Create domain test cases for integration testing*: design and generate test cases including test data that will be used for integration testing for domain and reused for integration testing for member products.

— *Derive domain test procedures for integration testing*: define the orders of test cases executed together. This task includes an ordered set of test cases to be re-executed due to changes.

— *Develop auxiliary test implementation for absent variants-related integration testing*: develop mocks or stubs and drivers necessary to execute test procedures related to absent variants.

7.2.4.3 Create domain test cases for integration testing

The goal of this task is to design test inputs such as domain test cases, test data and domain test scenarios to be used for domain integration testing and to be reused for application integration testing. Regression testing may be required when variability is newly added or changed, so this task includes test case selection.

The method should support creating domain test cases for integration testing with the following capabilities:

— addressing variability in integrated units for generating domain test inputs;

— providing appropriate variability mechanisms to be used for integration test cases;

— representing binding time explicitly to control the testability of integrated units;

— supporting relation management between variability in domain integration test inputs and relevant integrated domain units for test evolution in accordance with changes on variability in integrated units; and

— providing ways for test selection for re-testing integrated domain units.

A tool should support creating domain test cases for integration testing by allowing the user to do the following:

— utilize integration test cases repository;

— use notation for variability in domain test cases (including test data);

— document domain test cases (including test data); and

— perform automated test selection for domain integration testing.

7.2.4.4 Derive domain test procedures for integration testing

The goal of this task is to collect integration test clues and then to define types of required domain integration testing.

The method should support deriving domain test procedures for integration testing with the following capabilities:

— providing variability treatment approaches for a variability related set of integration test cases; and

— providing variability mechanisms to be used in integration test procedures.

A tool should support deriving domain test procedures for integration testing by allowing the user to do the following:

— utilize notation (description) for variability in integration test procedures.

7.2.4.5 Develop auxiliary test implementation for absent variants-related integration testing

The goal of this task is to select domain- or application-specific test inputs such as test cases, test data and test scenarios to be re-executed.

The method should support developing auxiliary test implementation for absent variants-related integration testing with the following capabilities:

— providing proper treatment for implementing mocks or stubs and drivers for executing variability relevant integration test cases and test procedures; and

— providing appropriate approaches for reusing mocks or stubs and drivers.

A tool should support developing auxiliary test implementation for absent variants-related integration testing by allowing the user to do the following:

— utilize environment for implementing mocks or stubs and drivers for variability; and

— reuse auxiliary test implementations.

7.2.5 Domain test design for system testing

7.2.5.1 General

This activity addresses the domain test design for system testing.

7.2.5.2 Tasks

The organization shall implement the following tasks with respect to the domain test design for system testing activity.

— *Create domain test cases for system testing*: design and generate system test cases including test data that will be used for system testing for domains and reused for system testing for member products.

— *Derive domain test scenarios for system testing*: define the orders of system test cases executed together. This task includes an ordered set of test cases including variability to be re-executed due to changes.

7.2.5.3 Create domain test cases for system testing

The goal of this task is to design test inputs such as domain test cases, test data and domain test scenarios to be used for system testing in domain engineering and to be reused for application testing depending on product line test strategy. Regression testing may be required when variability is newly added or changed, so this task includes test case selection.

The method should support creating domain test cases for system testing with the following capabilities:

— addressing variability in domain test inputs;

— providing appropriate variability mechanisms to be used for generating system test cases;

— representing binding time explicitly to control testability;

— supporting relation management between variability in domain system test inputs and relevant domain requirements for test evolution in accordance with changes on variable requirements; and

— providing ways for test selection for regression testing.

A tool should support creating domain test cases for system testing by allowing the user to do the following:

— utilize system test cases repository;

— use notation for variability in domain system test cases (including test data);

— document domain system test cases (including test data); and

— perform automated test selection for system testing in domain engineering.

7.2.5.4 Derive domain test scenarios for system testing

The goal of this task is to collect system test clues and then to define types of required system testing in domain engineering.

The method should support deriving domain test scenarios for system testing with the following capabilities:

— providing variability treatment approaches for a variability-related set of system test cases; and

— providing variability mechanisms used in system test procedures.

A tool should support deriving domain test scenarios for system testing by allowing the user to do the following:

— utilize notation (description) for variability in system test procedures.

7.3 Domain test environment set-up and maintenance

7.3.1 Principal constituents

7.3.1.1 Purpose

The purpose of this sub process is to provide integrated and interoperable environments for domain/application testing within and between domain and application. This process supports interoperability among tools that support domain and application process including tools for variability management.

7.3.1.2 Inputs

The following inputs should be available to perform the domain test environment set-up and maintenance process:

— environment (e.g. tools, equipment) requirements for domain test;

— interoperability requirements in domain test;

— tool information used in a product line; and

— change requests for domain test environments.

7.3.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the domain test environment set-up and maintenance process.

— *Enabling environments for supporting domain test* are established.

— *Integrated domain and application test environments* are established.

7.3.1.4 Tasks

The organization shall implement the following tasks with respect to the domain test environment set-up and maintenance process.

— *Set up domain test environments*: implement or set up domain test environment including hardware, software, test data and test tools.

— *Enable interoperability with other domain engineering environments*: allow an interoperable environment between domain testing and the relevant domain process in order to access domain artefacts to test. This also allows interoperation environments between variability models in testing and other relevant domain processes.

— *Maintain domain test environments*: change the domain test environments and share the status with the relevant stakeholders.

7.3.2 Set up domain test environments

The goal of this task is to set up and maintain test environments that support domain testing.

The method should support setting up domain test environments with the following capability:

— providing proper treatments in test for various binding times of large amount of variability.

A tool should support setting up domain test environments by allowing the user to do the following:

— utilize domain test environments that can treat variability among others; and

— refer install and uninstall information for an environment.

7.3.3 Enable interoperability with other domain engineering environments

The goal of this task is to provide enabling interoperable environments for proceeding domain testing from domain engineering artefacts.

The method should support enabling interoperability with other domain engineering environments with the following capabilities:

— supporting interoperation between domain test process and relevant domain process; and

— providing ways to link domain test artefacts and variability models.

A tool should support enabling interoperability with other domain engineering environments by allowing the user to do the following:

— interoperate with domain artefacts produced in the test relevant domain process;

— exchange required data for proceeding domain test; and

— link variability models to related domain test artefacts.

7.3.4 Maintain domain test environments

The goal of this task is to set up and maintain test environments that support domain testing.

The method should support maintaining domain test environments with the following capabilities:

— allowing maintenance for domain test environments among others.

A tool should support maintaining domain test environments by allowing the user to do the following:

— utilize integrated maintenance environments for domain test environments with other environments; and

— refer install and uninstall information for an environment.

7.4 Domain test execution

7.4.1 Principal constituents

7.4.1.1 Purpose

The purpose of this sub process is to execute tests for domain assets. In the middle of domain engineering, components are loosely coupled with each other, so there is testing in the absence of functioning components. This sub process should address how to deal with those assets in accordance with the established product line test strategy.

Test execution for absent variants can be performed during domain engineering by adding test codes for absent variants. In addition, test execution may be performed during application engineering when variants are available. A test case can be executed before or after variability binding in products (in an extreme case it can be executed after all variabilities are resolved), and the bindings can occur during development, compiling, linking or run time.

7.4.1.2 Inputs

The following inputs should be available to perform the domain test execution process:

— domain assets to be tested;

— domain test inputs; and

— variability models in test.

7.4.1.3 Outcomes

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

— *Domain test results for variability-relevant tests* are recorded.

7.4.1.4 Activities

The organization shall implement the following activities with respect to the domain test execution process:

— domain static testing;

— domain dynamic test execution.

7.4.2 Domain static testing

7.4.2.1 General

The goal of this activity is to review and inspect domain requirements, architecture, detailed design and codes in the case of software. Domain static testing includes architectural texture and domain variability models.

7.4.2.2 Tasks

The organization shall implement the following tasks with respect to the domain static testing activity.

— *Prepare review, inspection or static analysis for domain*: prepare materials (e.g. checklists) necessary to conduct domain static testing.

— *Execute domain static test*: run static testing by discriminating test execution for the commonality and variability so as to share what has been tested or what has not.

— *Record static test results*: document test execution and its test results by differentiating the test results for commonality from those for variability so as to share what has passed or what has not.

7.4.2.3 Prepare review, inspection or static analysis for domain

The goal of this task is to prepare for initiating domain static test.

The method should support preparing review, inspection or static analysis for domain with the following capabilities:

— defining domain static test inputs for variability; and

— allowing reusability of domain static test inputs.

A tool should support preparing review, inspection, or static analysis for domain by allowing the user to do the following:

— share domain static test inputs for variability with the related test engineers of member products.

7.4.2.4 Execute domain static test

The goal of this task is to conduct domain static test by using prepared domain static test inputs.

The method should support executing domain static test for domain with the following capability:

— supporting details (e.g. “how” to execute) for executing domain static test.

A tool should support executing domain static test by allowing the user to do the following:

— allowing automatic analysis based on the defined domain static test methods.

7.4.2.5 Record static test results

The goal of this task is to document the entire results of domain static test.

The method should support recording static test results with the following capability:

— providing templates for recording domain static test results.

A tool should support recording static test results by allowing the user to do the following:

— generate test result records.

7.4.3 Domain dynamic test execution

7.4.3.1 General

The goal of this activity is to carry out domain testing with executable implementations (e.g. test levels such as unit, integrations, system and acceptance testing in software or test types such as performance testing, security testing and so on).

7.4.3.2 Tasks

The organization shall implement the following tasks with respect to the domain dynamic test execution activity.

— *Execute domain test procedures*: run an ordered set of test cases.

— *Compare domain test results*: determine the pass/fail of domain testing. The results should be compared by discriminating test execution for the commonality and variability.

— *Record variability relevant test execution*: document test execution and test results related to variability so as to be referred by product line members.

7.4.3.3 Execute domain test procedures

The goal of this task is to run test cases or test suites according to test procedures.

The method should support executing domain test procedures with the following capability:

— providing treatment approaches for executing test procedures related to variability.

A tool should support executing domain test procedures by allowing the user to do the following:

— support (semi-)automatic execution of test procedures related to variability.

7.4.3.4 Compare domain test results

The goal of this task is to compare domain test results with the expected test results.

The method should support comparing domain test results with the following capability:

— providing test results comparison approaches for test procedures including test executions for variability.

A tool should support comparing domain test results by allowing the user to do the following:

— perform (semi-)automatic comparison considering variability relevant test execution.

NOTE Capabilities such as Capture and playback, Test comparator and Debugging used in single system test can be used as they are. However, those capabilities are adapted by considering the existence of variability.

7.4.3.5 Record variability relevant test

The goal of this task is to document tested and untested domain artefacts for avoiding redundant testing in application testing.

The method should support recording variability relevant test with the following capabilities:

— maintaining information on whether domain artefacts have been tested or not; and

— defining attributes necessary to record variability-relevant test.

A tool should support recording variability relevant test by allowing the user to do the following:

— record and report variability-relevant testing status.

7.5 Domain test reporting

7.5.1 Principal constituents

7.5.1.1 Purpose

The purpose of this sub process is to support integrated management for domain testing. The overall progress and status of domain testing are monitored and controlled. Defects in domain assets and feedback from application test are also managed to the closure.

7.5.1.2 Inputs

The following inputs should be available to perform the domain test reporting process:

— product line test strategy;

— product line test processes;

— domain test requirements;

— defects for domain assets reported from application testing; and

— status reports for domain testing.

7.5.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the domain test reporting process.

— *Domain test status reports including information for tested, partially tested or untested domain assets* are produced.

— *Defects in domain artefacts* are traced and managed.

7.5.1.4 Tasks

The organization shall implement the following tasks with respect to the domain test reporting process.

— *Analyse domain test results*: trace the status of defects in both commonality and variability reported during application testing as well as domain testing.

— *Create/update domain test reports*: produce reports on domain test status and results.

7.5.2 Analyse domain test results

The goal of this task is to manage the test execution status especially the status of variabilities that are tested or not. Defects with their status are also traced.

The method should support analysing domain test results with the following capability:

— tracing the test status of variabilities.

A tool should support analysing domain test results by allowing the user to do the following:

— collect and summarize the domain test execution status.

7.5.3 Create/update domain test reports

The goal of this task is to generate the reports for test results including the status of defects.

The method should support creating/updating domain test reports with the following capability:

— defining domain test report items.

A tool should support creating/updating domain test reports by allowing the user to do the following:

— summarize domain test results for reporting; and

— utilize the domain test report template.

8 Asset management in testing

8.1 General

The major assets in testing comprise: test cases, test data, test scenarios and test mocks or stubs. Domain test artefacts aforementioned are designed to be reused in application testing, so they should be well-structured in order to support the right and efficient reuse of test assets. Structuring test assets shall follow the structure defined in asset management of ISO/IEC 26555. Asset management in testing process includes the following two key sub processes.

— *Domain test artefacts as domain assets* structures domain test artefacts that will be reused by the product line members.

— *Application test artefacts as application assets* structures application test artefacts that will be revisited due to the product line or application evolution.

8.2 Domain test artefacts as domain assets

8.2.1 Principal constituents

8.2.1.1 Purpose

The purpose of this sub process is to make domain test artefacts into reusable asset configuration. The configuration should comply with that defined in asset management.

8.2.1.2 Inputs

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

— domain test artefacts;

— application test assets that have reuse potential; and

— requests from an application engineer to make test artefacts as envisioned domain assets.

8.2.1.3 Outcomes

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

— *Configuration and annotation for domain test artefacts* are established.

8.2.1.4 Tasks

The organization shall implement the following tasks with respect to the domain test artefacts as domain assets process.

— *Identify domain test artefacts managed as domain assets*. Domain test artefacts such as test models, test cases, scenarios and test data that will be managed as domain assets are identified. This task also covers application test assets that have reuse potential.

— *Structure the configuration and annotation of domain test artefacts*. Domain requirements artefacts identified as domain assets are structured according to the defined domain asset configuration and attached annotations for proper reuse.

8.2.2 Identify domain test artefacts managed as domain assets

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

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

— selecting domain test artefacts that have reuse potential;

— evaluating selected domain test artefacts (e.g. checklist for evaluating reusability); and

— establishing backward trace links with domain assets at the relevant domain level.

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

— importing domain test artefacts that have reuse potential; and

— storing the list of domain test artefacts as domain assets.

8.2.3 Structure configuration and annotation for domain test assets

The goal of this task is to define and develop the further structure of domain test artefacts useful for reuse at the application developments.

The method should support structuring configuration and annotation with the following capabilities:

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

— identifying annotations' necessity to reuse domain test artefacts as domain assets during the application testing; and

— validating configuration and annotations for domain test artefacts.

A tool should support structuring configuration and annotation with the following capabilities:

— accessing the existing information for defining configuration and annotation (the configuration includes trace links with the relevant development assets);

— accessing domain requirements artefacts; and

— utilizing an editor for template definition.

8.3 Application test artefacts as application assets

8.3.1 Principal constituents

8.3.1.1 Purpose

The purpose of this sub process is to establish the structure of application assets in requirements as constituent relationships with other application engineering processes.

8.3.1.2 Inputs

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

— application test artefacts.

8.3.1.3 Outcomes

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

— *Configuration and annotations for application test artefacts are defined or developed.*

8.3.1.4 Tasks

The organization shall implement the following tasks with respect to the application test artefacts as application assets process.

— *Identify application test artefacts managed as application assets.* Application test artefacts that have relationships with successive application engineering processes are selected and determined.

— *Structure configuration and annotation for application test assets.* The structure and annotations of application test assets that are necessary to maintain them within the asset repository or to reuse them at each application development are defined and developed.

8.3.2 Identify application test artefacts managed as application assets

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

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

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

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

— importing application test artefacts; and

— reviewing and editing application test artefacts.

8.3.3 Structure configuration and annotation for application test assets

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

The method should support structuring configuration and annotation for application test assets with the following capabilities:

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

— identifying annotations required to reuse the application requirements asset in application developments; and

— validating configuration and annotations for application requirements assets.

A tool should support structuring configuration and annotation for application test assets with the following capabilities:

— accessing the existing information for defining configuration and annotation;

— accessing application test assets;

— utilizing an editor for template definition;

— establishing trace links among application test assets; and

— establishing trace links between assets resulting from the later application engineering processes.

9 Variability management in testing

9.1 General

Variability should be managed in product line testing as like in other product line lifecycle processes. Test artefacts such as test plans, test cases, test data and test scenarios may include variability that is spread across the different artefacts. Variability management in testing process includes the following three key sub processes.

— *Variability mechanism category in testing* maintains variability mechanism category suitable for variability bound at the testing phase.

— *Variability in test artefacts* serves to express and document variabilities included in domain and application test artefacts.

— *Traceability of variability in test* manages trace links of variability in test artefacts with separately defined variability models.

9.2 Variability mechanism category in testing

9.2.1 Principal constituents

9.2.1.1 Purpose

The purpose of this sub process is to establish and maintain variability mechanism category specific to testing.

Variability mechanisms in testing used for designing test artefacts differ in accordance with the applied SSPL test strategy (for example, commonality and reuse strategy and sample application strategy), test artefacts (for example, test plans, test scenarios and test cases) and test levels (for example, unit test, integration test and system test). Variability mechanisms should be identified by considering the relationships between the implementation under test and those mentioned above, i.e. test strategy, test artefacts and test levels. ISO/IEC 26557 defines variability mechanism categories as SSPL specifically defined mechanisms, Language extension mechanisms and Language support mechanisms. The following are the exemplar variability mechanism categories in testing.

— *SSPL specifically defined mechanism:* "Extensions" row and tags in Product Line Use Cases for system testing, segmentation and fragmentation mechanisms for test case scenarios by sequence diagram;

— *Language extension mechanism:* UML sequence diagram for domain system and integration test scenario (note with tagged values) and UML activity diagram for test model (extended decision point with notes); and

— *Language support mechanism:* UML sequence diagram with genericity marks; parameters, wildcards.

9.2.1.2 Inputs

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

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

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

9.2.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the variability category in testing process.

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

— *Guidance is tailored specific to testing phase*.

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

9.2.1.4 Tasks

The organization shall implement the following tasks with respect to the variability category in testing process.

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

— *Guide the use of variability mechanism category by PL test strategy*: support correct use of variability mechanisms in accordance with the selected product line test strategy.

— *Guide the use of variability mechanism category by test levels*: support correct use of variability mechanisms in accordance with each test level.

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

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

9.2.2 Identify variability mechanisms in testing by category

The goal of this task is to find variability mechanisms suitable for the established product line test strategy and test levels; unit test, integration test and system test. Variability mechanisms should also be reviewed on test artefacts and test targets.

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

— reviewing the available variability mechanisms whether they comply with the explored architectural decisions and textures in the testing phase;

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

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

— evaluating ease of binding.

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

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

— store rationales for variability mechanism selection.

9.2.3 Guide the use of variability mechanism category by PL test strategy

The goal of this task is to provide guidance for assessing, selecting and deploying variability mechanisms depending on the established PL test strategy based on the variability mechanism category.

In the case of commonality and reuse strategy, variation points are implemented in test artefacts and decisions for their binding times depending on the bindings of implementation under test are also required. Whereas, sample application strategy selects a minimum number of representative member products from a product line. This strategy assumes that testing for commonalities has been completed by testing the selected member products. As for variability, sample application strategy helps ensure whether variability mechanisms work correctly.

The overall guidance is provided by ISO/IEC 26557. This task supports concrete guides filled with test-specific practices. Annex B provides exemplified test execution recommendations for each of the variability mechanisms.

The method should support guiding the use of variability mechanism category by PL test strategy with the following capabilities:

— tailoring variability mechanism selection criteria specific to the selected PL test strategy;

— tailoring detailed procedures for variability mechanism selection and use specific to the selected PL test strategy; and

— tailoring configuration guides and rules specific to the selected PL test strategy.

A tool should support guiding the use of variability mechanism category by PL test strategy by allowing the user to do the following:

— share guides and rules for variability mechanism operation by PL test strategy with relevant participants; and

— perform instant access for guides and rules for performing variability mechanism operation by PL test strategy.

9.2.4 Guide the use of variability mechanism category by test levels

The goal of this task is to provide detailed guidance for assessing, selecting and deploying variability mechanisms in accordance with test levels.

In case of unit testing the possible number of variants for a variation point is controllable, so unit testing for domain can be conducted during domain testing. The test artefacts for domain unit testing and ways to validate variability mechanisms can differ from variability mechanisms used in the test targets.

In integration testing, the number of possible combination of variants within components to be integrated can be uncontrollable. In such cases, variation points are encoded into test artefacts for domain integration testing. Thus, proper variability mechanisms for testing should be selected in accordance with the variability mechanisms used in test targets, and the binding times of the implemented variation points within test artefacts should be decided depending on the variability binding times of the relevant test target.

In case of system testing, test artefacts can be generated during domain engineering to be reused by member products, so the test artefacts include variabilities. In such cases, ways to use variability mechanisms are similar to those in integration testing.

New variabilities can be added during testing to allow for differences in testing among member products. Newly added variabilities should also be realized in test artefacts and they can be bound at the same test level or after where they are introduced.

The method should support guiding the use of variability mechanism category by test levels with the following capabilities:

— tailoring variability mechanism selection criteria specific to each test level;

— tailoring detailed procedures for variability mechanism selection and use specific to each test level; and

— tailoring configuration guides and rules specific to each test level.

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

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

— access guides and rules for performing variability mechanism operation by test levels.

9.2.5 Trace the usage status of variability mechanism category in testing

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

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

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

— confirming variability mechanisms in testing whether they can correctly realize the defined variability dependencies;

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

— providing evaluation algorithms for verifying whether a variability mechanism confirms the defined variability dependencies and constraints; and

— verifying the testability of variability mechanisms used in testing.

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

— visualize variability mechanisms in testing;

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

— perform the testability verification of variability mechanisms used in testing.

9.2.6 Update variability mechanism category in testing

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

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

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

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

— preparing items for variability mechanism category improvement;

— improving the variability mechanism category; and

— propagating improvement results to the variability mechanism pool.

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

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

— share the improved variability mechanism category; and

— use automatic propagation of improvement results to the variability mechanism pool.

9.3 Variability in test artefacts

9.3.1 Principal constituents

9.3.1.1 Purpose

The purpose of this sub process is to determine how to generate domain test artefacts that include variabilities thereafter to support reuse in test artefacts for member products of SSPL.

Domain test cases can be generated in a form that preserves variation points, which have to be resolved at a later stage, or can be separately generated for each set of variants for the variability. Meanwhile, testing in application engineering reuses domain test cases across different products in the product line. In case the domain test cases include variants, they are transformed into application test cases through variability binding. Otherwise, a mechanism for selecting test cases related to a specific application is necessary.

9.3.1.2 Inputs

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

— product line test strategy;

— variability models of both domain engineering and application engineering; and

— domain artefacts to be tested.

9.3.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the variability in the test artefacts process.

— *Variability types in test artefacts* are defined.

— *Ways to describe variability in test artefacts* are defined.

— *Binding mechanism for domain test scenarios and test cases* are defined.

9.3.1.4 Tasks

The organization shall implement the following tasks with respect to the variability in test artefacts process.

— *Define variability type in test artefacts*: define possible types of variability in test cases, test data and test procedures.

— *Define variability representation in test artefacts*: define ways to represent variability in test cases, test data and test procedures in the forms of easily reused and bound in application testing.

9.3.2 Define variability type in test artefacts

The goal of this task is to define variability types that test cases and test scenarios can include. Test data related to variability should also be covered. An example of a variability type in test artefacts can be an optional or alternative test case in test sequence.

The method should support defining variability type in test artefacts with the following capabilities:

— defining possible variability types in test artefacts; and

— verifying whether the defined variability types in test artefacts are mutually exclusive, complete and correct.

A tool should support defining variability type in test artefacts by allowing the user to do the following:

— perform verification for the defined variability types.

9.3.3 Define variability representation in test artefacts

The goal of this task is to describe and document variabilities in test cases and test scenarios for further reuse.

The method should support defining variability representation in test artefacts with the following capabilities:

— defining notation for describing variability types in test artefacts;

— defining ways to represent variability in test artefacts; and

— defining documentation components for variability in test artefacts.

A tool should support defining variability representation in test artefacts by allowing the user to do the following:

— perform variability modelling for the defined variability types in test artefacts;

— conduct variability description in test artefacts including binding time; and

— utilize a documentation template including the defined documentation components.

9.4 Traceability of variability in test

9.4.1 Principal constituents

9.4.1.1 Purpose

The purpose of this sub process is to establish and maintain trace links between variability models and domain test artefacts.

9.4.1.2 Inputs

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

— variability models in test; and

— domain test artefacts.

9.4.1.3 Outcomes

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

— *Trace links between variability model and domain test assets* are established and maintained.

— *Trace links between application variability model and domain test assets* are established and maintained.

9.4.1.4 Tasks

The organization shall implement the following tasks with respect to the traceability of variability in test process.

— *Define explicit links between variability in test assets and variability model*: establish and maintain trace links between test assets related to variability and the variability model.

— *Define explicit links between application test assets and application variability model*: establish and maintain trace links between test assets used in a member product and its variability model.

9.4.2 Define explicit links between variability in test assets and variability model

The goal of this task is to establish and maintain explicit links between variability in test assets and domain variability model.

The method should support defining explicit links between variability in test assets and variability model with the following capability:

— determining the links between variability model and variability in test artefacts.

A tool should support defining explicit links between variability in test assets and variability model by allowing the user to do the following:

— visualize trace links between test artefacts and the variability model; and

— record links between variability in the test and the variability model.

9.4.3 Define explicit links between application test assets and application variability model

The goal of this task is to establish and maintain explicit trace links between variability defined in a member product and application variability model.

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

— determining the links of variability model with variability in test artefacts.

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

— visualize trace links between application test artefacts and application variability model; and

— record links between application-specific variability in test and application variability model.

10 Application testing

10.1 General

Application engineering is directly relevant to customer products and often needs to deal with changes in customer needs. Domain artefacts may not satisfy the specific product's needs completely. Thus, there are gaps between what is available and what is required. Unsatisfied needs may be met by implementing application-specific artefacts or by adapting domain artefacts to fill the gaps. Where application-specific artefacts are developed, they are tested at each application. When variability bindings are conducted or application-specific artefacts are integrated into the existing artefacts, regression testing shall be performed as necessary.

Application testing process includes the following four key sub processes.

— *Application test initiation and design* generates application-specific test cases and test procedures.

— *Application test environment set-up and maintenance* sets up and maintains application test environments. Application test environments shall be composed in detail based on test environments described in the test plan, defined as a result of the product line test planning process.

— *Application test execution* performs application-specific dynamic and static testing based on the test procedures generated during application test design and implementation.

— *Application test reporting* describes the results of identified failures and its locations.

10.2 Application test initiation and design

10.2.1 Principal constituents

10.2.1.1 Purpose

The purpose of this sub process is to plan and prepare for application testing focusing on application-specific parts that are not covered in domain testing. The test plan should address the reuse and the test scope that will be reused or covered during application testing. Test inputs for application-specific parts should be newly developed.

Application test cases may be created from application artefacts or through application test models. Application test cases may be created either directly from domain test cases by using binding information of application artefacts or through test case selection. Application-specific test cases are created and executed during application engineering. Regression testing is used in application testing to avoid redundant testing with domain testing.

10.2.1.2 Inputs

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

— organizational test plans;

— application test references (e.g. application requirements specification and application architecture specification); and

— application variability models in test.

10.2.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the application test initiation and design process.

— *Application test requirements and conditions* are defined.

— *Application test cases* are generated.

— *Application test procedures* are defined.

10.2.1.4 Activities

The organization shall implement the following activities with respect to the application test initiation and design process:

— application test initiation;

— application-specific test design for unit testing;

— application test design for integration testing; and

— application test design for system testing.

10.2.2 Application test initiation

10.2.2.1 Tasks

The organization shall implement the following tasks with respect to the application test initiation activity.

— *Initiate application testing*: set up readiness for proceeding application-specific testing.

— *Define application test requirements and conditions*: identify test requirements to be satisfied and conditions for the application-specific test items.

10.2.2.2 Initiate application testing

The goal of this task is to develop application test plans that include application-specific objectives, the scope of the application test, responsibilities, methods, tools to be applied, schedules for application tests and test completion criteria.

The method should support initiating application testing with the following capabilities:

— defining entry criteria to be assured in order to perform the application test;

— defining entry criteria for the application test in order to perform the application test;

— defining guidelines to evaluate readiness;

— selecting artefacts to be evaluated to confirm to the readiness; and

— defining documentation components and guides for documentation.

A tool should support initiating application testing by allowing the user to do the following:

— access the artefacts to be evaluated; and

— share the defined entry criteria with the relevant participants.

10.2.2.3 Define application test requirements and conditions

The goal of this task is to establish test requirements that should be tested in application-specific testing.

The method should support defining application test requirements and conditions with the following capabilities:

— defining the form of test requirements to be used for deriving application test cases and application test procedures;

— analysing the characteristics of application-specific testing in accordance with the nature of a member product; and

— refining test architecture and test coverage in accordance with the application-specific test requirements.

A tool should support defining application test requirements and conditions by allowing the user to do the following:

— access sources for test design such as application-specific requirements specifications, application architecture, application-specific detailed design specifications and parts/codes;

— utilize modelling notation for test requirements; and

— utilize modelling notation for test architecture.

10.2.3 Application-specific test design for unit testing

10.2.3.1 General

This activity addresses application-specific test design for unit testing.

10.2.3.2 Tasks

The organization shall implement the following tasks with respect to the application-specific test design for unit testing activity.

— *Create application-specific test cases for unit testing*: select domain test assets and generate new test cases including test data that will be used to application-specific unit testing.

— *Derive application-specific test procedures for unit testing*: define the orders of test cases executed together by selecting domain unit test cases and application-specific unit test cases. This task includes an ordered set of test cases to be re-executed in accordance with bindings.

— *Develop auxiliary test implementation for application-specific unit testing*: develop mock or stub necessary to execute test procedures for application-specific units.

10.2.3.3 Create application-specific test cases for unit testing

The goal of this task is to generate application-specific test cases and test data for unit testing. Regression testing may be required in accordance with binding results.

The method should support creating application-specific test cases for unit testing with the following capabilities:

— providing test selection approaches considering variability binding results;

— generating application-specific test cases from test items that include un-bound variability; and

— providing ways for test selection for regression testing in application testing by considering application-specific variability.

A tool should support creating application-specific test cases for unit testing by allowing the user to do the following:

— document application-specific test cases (including test data); and

— perform automated test selection in application testing considering variability binding.

10.2.3.4 Derive application-specific test procedures for unit testing

The goal of this task is to collect test clues and then to define types of required application testing.

The method should support deriving application-specific test procedures for unit testing with the following capabilities:

— controlling the number of test cases to be re-executed in accordance with bindings;

— adding application-specific test cases to derived test procedures; and

— providing ways for test selection for regression testing in application testing by considering binding results.

A tool should support deriving application-specific test procedures for unit testing by allowing the user to do the following:

— perform automated test selection in application testing by considering binding results; and

— perform automated test selection in application testing by considering application-specific variability.

10.2.3.5 Develop auxiliary test implementation for application-specific unit testing

The goal of this task is to implement mocks or stubs to handle absent variants.

The method should support developing auxiliary test implementation for application-specific unit testing with the following capabilities:

— supporting the reuse of domain auxiliary test implementation for developing application-specific auxiliary test implementation;

— supporting the development of application-specific mocks or stubs; and

— providing appropriate approaches for dealing with the diverse binding times of variability in units.

A tool should support developing auxiliary test implementation for application-specific unit testing by allowing the user to do the following:

— utilize an environment for reusing domain auxiliary test implementation; and

— develop auxiliary test implementation for application-specific units.

10.2.4 Application test design for integration testing

10.2.4.1 General

This activity addresses test design for application integration testing. Application integration testing reuses domain test assets in accordance with variability binding results and adds new application-specific test assets for a member product. Proper domain test assets for integration testing are selected and reused as they are, or they are modified if necessary.

10.2.4.2 Tasks

The organization shall implement the following tasks with respect to the application test design for integration testing activity.

— *Create application test cases for integration testing*: select domain test assets and generate new test cases including test data that will be used to application-specific integration testing.

— *Derive application test procedures for integration testing*: define the orders of test cases executed together by selecting domain integration test cases and application-specific integration test cases. This task includes an ordered set of test cases to be re-executed in accordance with bindings.

— *Develop auxiliary test implementation for application integration testing*: develop the mock or stub necessary to execute test procedures for application-specific integrated units.

10.2.4.3 Create application test cases for integration testing

The goal of this task is to generate application-specific test cases and test data. Regression testing may be required in accordance with binding results.

The method should support creating application test cases for integration testing with the following capabilities:

— providing test selection approaches considering variability binding;

— generating application-specific test cases from test items that include un-bound variability; and

— providing test selection methods for regression testing in application testing by considering application-specific variability.

A tool should support creating application test cases for integration testing by allowing the user to do the following:

— document application-specific test cases (including test data); and

— perform automated test selection in application testing considering variability binding.

10.2.4.4 Derive application test procedures for integration testing

The goal of this task is to collect test clues and then to define types of required application testing.

The method should support deriving application test procedures for integration testing with the following capabilities:

— controlling the number of test cases to be re-executed in accordance with the criteria; and

— providing test selection methods for regression testing in application testing by considering binding results.

A tool should support deriving application test procedures for integration testing by allowing the user to do the following:

— perform automated test selection in application testing by considering binding results; and

— perform automated test selection in application testing by considering application-specific variability.

10.2.4.5 Develop auxiliary test implementation for application integrated testing

The goal of this task is to implement mocks or stubs to handle absent variants.

The method should support developing auxiliary test implementation for application integration testing with the following capabilities:

— providing proper treatment for auxiliary test implementation for executing absent variants-relevant test cases and test procedures; and

— providing appropriate approaches for reusing auxiliary test implementation.

A tool should support developing auxiliary test implementation for application integration testing by allowing the user to do the following:

— use an environment for auxiliary test implementation for absent variants; and

— reuse auxiliary test implementation.

10.2.5 Application test design for system testing

10.2.5.1 General

This activity addresses application test design for system testing. In most cases (except for domain-testing), only strategy application system testing is not conducted during domain engineering, so application system testing as the whole system is conducted for the first time. However, test assets for system testing are produced during domain requirements engineering, so domain test assets for system testing are reused with variability binding. Application-specific test assets can be added in accordance with application-specific requirements.

10.2.5.2 Tasks

The organization shall implement the following tasks with respect to the application test design for system testing activity.

— *Create application-specific test cases for system testing*: generate system test cases for a member product by binding domain system test cases and by adding new test cases for application-specific requirements.

— *Derive application-specific test procedures for system testing*: define the orders of test cases executed together. This task includes an ordered set of test cases to be re-executed in accordance with bindings.

10.2.5.3 Create application test cases for system testing

The goal of this task is to create application-specific test cases and test data for system testing. Regression testing may be required in accordance with binding results.

The method should support creating application test cases for system testing with the following capabilities:

— providing test selection approaches among domain test assets for system testing considering variability binding;

— generating application system test assets from domain test assets that include variability;

— creating application-specific system test assets for application-specific requirements; and

— providing test selection methods for regression testing for system testing by considering application-specific variability.

A tool should support creating application test cases for system testing by allowing the user to do the following:

— support documentation for application-specific system test cases (including test data); and

— support automated test selection for system testing considering variability binding.

10.2.5.4 Derive application test scenarios for system testing

The goal of this task is to collect test clues and then to define types of required application testing.

The method should support deriving application test scenarios for system testing with the following capabilities:

— controlling the number of test cases to be re-executed in accordance with the criteria; and

— providing ways for test selection for regression testing in system testing by considering binding results.

A tool should support deriving application test scenarios for system testing by allowing the user to do the following:

— perform automated test selection for system testing by considering binding results; and

— perform automated test selection for system testing by considering application-specific variability.

10.3 Application test environment set-up and maintenance

10.3.1 Principal constituents

10.3.1.1 Purpose

The purpose of this sub process is to provide required environments for application testing. This process supports interoperability among supporting tools for application process.

10.3.1.2 Inputs

The following inputs should be available to perform the application test environment set-up and maintenance process:

— application-specific environment (e.g. tools, equipment) requirements for application testing;

— interoperability requirements in application testing;

— tool information used in a product line; and

— change requests for application test environments.

10.3.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the application test environment set-up and maintenance process.

— *Enabling environments for supporting application test* are established.

— *Integrated domain and application test environments* are established.

10.3.1.4 Tasks

The organization shall implement the following tasks with respect to the application test environment set-up and maintenance process.

— *Set up application test environments*: implement or set up additional application-specific test environment on the established domain test environments.

— *Enable interoperability with other application engineering environments*: allow interoperation environment between application-specific testing and the relevant application process in order to refer application artefacts to perform testing.

— *Maintain application test environments*: change the domain test environments and share the status with the relevant stakeholders.

10.3.2 Set up application test environments

The goal of this task is to set up and maintain test environments required for specific member products.

The method should support setting up application test environments with the following capability:

— providing proper treatments in test for wide range of binding times of large amount of variability.

A tool should support setting up application test environments by allowing the user to do the following:

— use application test environments that can treat absent variants; and

— refer install and uninstall information for an environment.

10.3.3 Enable interoperability with other application engineering environments

The goal of this task is to provide enabling interoperable environments for application testing from domain testing and domain test artefacts.

The method should support enabling interoperability with other application engineering environments with the following capabilities:

— supporting interoperation between application test process and relevant domain process; and

— providing ways to link application test artefacts and application variability models.

A tool should support enabling interoperability with other application engineering environments by allowing the user to do the following:

— interoperate with application artefacts produced in the test relevant application process;

— exchange required data for performing application test; and

— link application variability models to related application test artefacts.

10.3.4 Maintain application test environments

The goal of this task is to manage changes on test environments that support application testing.

The method should support maintaining application test environments with the following capability:

— allowing maintenance for domain test environments among others.

A tool should support maintaining application test environments by allowing the user to do the following:

— use integrated maintenance environments for application-specific test environments with other environments; and

— refer install and uninstall information for an environment.

10.4 Application test execution

10.4.1 Principal constituents

10.4.1.1 Purpose

The purpose of this sub process is to execute tests for application focusing on application-specific parts and their relevant parts that are not covered in domain testing.

10.4.1.2 Inputs

The following inputs should be available to perform the application test execution process:

— application artefacts under testing;

— application test inputs; and

— application variability models.

10.4.1.3 Outcomes

The following outcome shall be available as a result of the successful implementation of the application test execution process.

— *Application test results* are recorded.

10.4.1.4 Activities

The organization shall implement the following activities with respect to the application test execution process:

— application static testing; and

— application dynamic test execution.

10.4.2 Application static testing

10.4.2.1 General

The goal of this activity is to review and inspect application-specific requirements, architecture, detailed design and codes in the case of software. Application static testing includes architectural texture and application variability models.

10.4.2.2 Tasks

The organization shall implement the following tasks with respect to the application static testing activity.

— *Prepare review, inspection or static analysis for application*: prepare materials (e.g. checklists) necessary to conduct application static testing.

— *Execute application static test*: run static testing for application-specific parts and for the selected variants.

— *Record static test results*: document test execution and its test results.

10.4.2.3 Prepare review, inspection or static analysis for application

The goal of this task is to prepare for initiating the application static test.

The method should support preparing review, inspection or static analysis for application with the following capabilities:

— defining static test inputs for application-specific parts;

— selecting reusable static test inputs; and

— allowing variability binding in static test inputs in accordance with variability binding in artefacts.

A tool should support preparing review, inspection or static analysis for application by allowing the user to do the following:

— utilize environments for static test input definition for application-specific parts;

— perform (semi-)automatic selection of reusable static test inputs; and

— perform (semi-)automatic variability binding in static test inputs in accordance with variability binding results in artefacts.

10.4.2.4 Execute application static test

The goal of this task is to conduct the application static test by using prepared application static test inputs.

The method should support executing static test for application-specific parts with the following capability:

— supporting details (e.g. “how” to execute) for executing the application static test.

A tool should support executing static test for application-specific parts by allowing the user to do the following:

— perform automatic analysis based on the defined application static test methods.

10.4.2.5 Record static test results

The goal of this task is to document the entire results of the application static test.

The method should support recording static test results with the following capability:

— providing templates for recording application static test results.

A tool should support recording static test results by allowing the user to do the following:

— record test results;

— trace the identified defects for variability relevant parts; and

— record environments in accordance with the defined template.

10.4.3 Application dynamic test execution

10.4.3.1 General

The goal of this activity is to carry out application-specific testing with executable implementations (e.g. test levels such as unit, integrations, system and acceptance testing in software or test types such as performance testing, security testing etc.).

10.4.3.2 Tasks

The organization shall implement the following tasks with respect to the application dynamic test execution activity.

— *Execute application test procedures*: run an ordered set of test cases.

— *Compare application test results*: determine the pass/fail of application testing and the results should be compared by discriminating test execution for the commonality and variability.

— *Record absent variants-relevant test execution*: document test execution and test results related to un-bound variability.

10.4.3.3 Execute application test procedures

The goal of this task is to perform application testing using application test cases and procedures.

The method should support executing application test procedures with the following capabilities:

— providing treatment approaches for absent variants;

— tracing the test status for application artefacts; and

— supporting maintenance of application test execution results.

A tool should support executing application test procedures by allowing the user to do the following:

— perform (semi-)automatic execution; and

— control test execution during automated test execution.

10.4.3.4 Compare application test results

The goal of this task is to compare test results and expected results for locating errors related to absent variants. Pass/fail for absent variants cannot be determined until they are bound, so comparison approaches should be able to handle this.

The method should support comparing application test results with the following capabilities:

— supporting definition of test oracles relevant to variability binding;

— supporting definition of test oracles for absent variants;

— providing test result judgement approaches including test executions for variability binding and absent variants; and

— allowing the integration of test results for commonality, bound variability and application-specific variability.

A tool should support comparing application test results by allowing the user to do the following:

— perform (semi-)automatic execution of test oracles relevant to variability binding;

— perform (semi-)automatic execution of test oracles for absent variants;

— perform (semi-)automatic judgement considering variability binding and absent variants; and

— record application test results by integrating test results for commonality, bound variability and application-specific variability.

NOTE Capabilities such as Capture and playback, Test comparator and Debugging used in single system test can be used as they are. However, those capabilities are adapted by considering absent variants.

10.4.3.5 Record absent variants-relevant test execution

The goal of this task is to record tested application artefacts related to absent variants so as to use the results after binding.

The method should support recording absent variants-relevant test execution with the following capabilities:

— maintaining test results related to absent variants;

— defining fields for record; and

— maintaining test asset information used to absent variants-relevant testing.

A tool should support recording absent variants-relevant test execution by allowing the user to do the following:

— record absent variants-relevant testing status in accordance with the defined fields; and

— trace test assets used for absent variants-relevant testing.

10.5 Application test reporting

10.5.1 Principal constituents

10.5.1.1 Purpose

The purpose of this sub process is to manage the progress and status of application testing.

10.5.1.2 Inputs

The following inputs should be available to perform the application test reporting process:

— application test requirements; and

— domain test plans.

10.5.1.3 Outcomes

The following outcomes shall be available as a result of the successful implementation of the application test reporting process.

— *Application test plans* are defined.

— *Application test status reports* are produced.

— *Defects in application artefacts* are traced and managed.

10.5.1.4 Tasks

The organization shall implement the following tasks with respect to the application test reporting process.

— *Analyse application test results*: trace the status of application-specific defects.

— *Create/update application test reports*: produce reports on application test status and results.

10.5.2 Analyse application test results

The goal of this task is to manage test execution status for application-specific artefacts and corresponding artefacts that should be retested.

The method should support analysing application test results with the following capabilities:

— supporting the analysis of identified defects together with domain or application phases where defects were introduced;

— determining whether the causes of defects are responsible for domain or application; and

— tracing application test results with domain or application phases analysed as defect sources.

A tool should support analysing application test results by allowing the user to do the following:

— collect and summarize application-specific test execution status;

— trace status of the analysed defects and their sources of domain or application engineering stages; and

— record identified defects with their introduced domain or application phases.

10.5.3 Create/update application test reports

The goal of this task is to generate the reports for test results including the status of defects.

The method should support creating/updating application test reports with the following capabilities:

— defining application-specific test report items; and

— providing application test report template in coordination with domain test report.

A tool should support creating/updating application test reports by allowing the user to do the following:

— summarize application test results for reporting;

— use electronic test report writing environment in accordance with the defined template; and

— share identified defect status with relevant participants.

Annex A
(informative)

Exemplar product line test strategy

A.1 Overview of testing strategy

The test strategy of SPL relates to how a domain test and an application test are carried out on the occasion of securing of reliability of the application in the product line. The test strategic decision depends on product properties (the number of the variations) and the product strategy (business model). Product properties and the point of view of the product strategy are covered in detail in Clause 5.

On a domain test and an application test, plural solutions exist in a test strategy to determine what kind of test allotment is carried out. The two strategies include Domain-testing-only strategy and Application-testing-only strategy.

— Domain-testing-only strategy: Domain Engineering organization tests all potential combinations of domain assets as a domain test.

— Application-testing-only strategy: Only the inspection of the basic function of domain assets is performed on a domain test, and an Application Engineering organization tests it as an application test about the variable setting result.

For a strategy to choose, the middle strategies of Domain-testing-only strategy and Application-testing-only strategy can normally be considered, because there are practical constraints such as resource, budget, early validation need and time to market.

A.2 Exemplar organizational test strategies

A.2.1 General

The following three middle strategies are suggested by Klaus Pohl, Günter Böckle and Frank van der Linden (see Reference [19]):

— SAS;

— CRS; and

— Combined SAS and CRS.

A.2.2 SAS

The SAS tests a few representative applications for assuring the quality of domain assets. When the test engineer chooses sample applications, each sample has one particular configuration. Through the tested sample applications, all common assets are tested under the selected variants of a sample application.

The SAS makes it possible to achieve an early validation of common assets; and it helps ensure that a qualified member product can be derived through the domain assets. Through selecting representative sample applications, major variabilities of the whole product line can also be validated. In addition, test assets are reused for testing the other member products.

However, the SAS does not focus on test asset reuse, so test assets produced during sample application testing do not have a proper structure that is easy to reuse, namely they do not have a proper structure

for their right selection. This means that those test assets do not have proper information to use for selecting test assets to test the remaining member products within a product line.

In the SAS, testing is performed in the same way as in the single application testing. Thus, lower learning efforts are required than other strategies.

A.2.3 CRS

The CRS tests available domain assets during domain engineering and produces reusable test assets for variable parts so that member products reuse those test assets for member product testing. The CRS produces domain test assets that can help ensure the systematic reuse of them in application testing as the other domain assets do.

The CRS enhances the reusability of domain test assets. Domain test assets (e.g. test cases, test scenarios and procedures) are selected (reused) in accordance with the configuration of a member product. However, many efforts can be required for creating and structuring domain test assets for easy reuse. In addition, this strategy should devise ways to avoid redundant testing.

A.2.4 Combined SAS and CRS

The combined SAS and CRS strengthen the advantage and make up for the weakness of each other. The CRS enforces the creation of reusable test assets in domain testing and the SAS realizes an early validation of a member product by performing testing for the fragment of a sample application. In this strategy, there is no complete member product, but the fragment of a sample application should be large enough to validate the derivation of an application.

Annex B
(informative)

Execution of SSPL testing

While test execution and the method of implementation of the variable mechanism affect the choice of the test tool, it is desirable for the concrete method of implementation of the variable mechanism to choose a test tool suitable for each. Some available ways are exemplified as follows.

— **Conditional compilation**

When #ifdef realizes variability by becoming it, core assets are tested by a unit test.

— **Component implementation**

When a component realizes variability, core assets are tested by a combination test.

— **Component calls**

When architecture realizes variability, core assets are tested by system test.

Annex C
(informative)

Mapping of ISO/IEC/IEEE 29119-2 and ISO/IEC/IEEE 15288

Table C.1 — Mapping of ISO/IEC/IEEE 29119-2, ISO/IEC/IEEE 15288 and ISO/IEC 26554

Area	ISO/IEC/IEEE 29119-2		ISO/IEC/IEEE 15288		ISO/IEC 26554	
	Clause#	Clause name	Clause#	Clause name	Clause#	Clause name
Organizational Test Process	6.2	Organizational Test Process	6.2.1	Life Cycle Model Management Process	6.2 6.3	Product Line Test Strategy Product Line Test Process
Test Management Processes	7.2	Test Planning Process	6.2.3	Project Portfolio Management Process	6.4	Product Line Test Planning
			6.3.1	Project Planning Process		
			6.3.4	Risk Management Process		
			6.4.6	Verification Process		
			6.4.8	Validation Process		
	7.3	Test Monitoring and Control Process	6.2.3	Project Portfolio Management Process	6.5	Product Line Test Monitoring and Control
			6.3.2	Project Assessment and Control Process		
			6.3.4	Risk Management Process		
			6.3.7	Measurement Process		
			6.4.6	Verification Process		
			6.4.8	Validation Process		
	7.4	Test Completion Process	6.2.3	Project Portfolio Management Process	6.5.5 7.5 10.5	Report Product Line Test Progress Domain Test Reporting Application Test Reporting
			6.3.2	Project Assessment and Control Process		
			6.3.4	Risk Management Process		
			6.3.7	Measurement Process		

Table C.1 *(continued)*

Area	ISO/IEC/IEEE 29119-2		ISO/IEC/IEEE 15288		ISO/IEC 26554	
	Clause#	Clause name	Clause#	Clause name	Clause#	Clause name
Dynamic Test Processes	8.2	Test Design & Implementation Process	6.4.8	Validation Process	7.2 10.2	Domain Test Initiation and Design Application Test Initiation and Design
	8.3	Test Environment Set-Up & Maintenance Process	6.4.8	Validation Process	7.3 10.3	Domain Test Environment Set-Up and Maintenance Application Test Environment Set-Up and Maintenance
	8.4	Test Execution Process	6.4.5	Integration Process	7.4 10.4	Domain Test Execution Application Test Execution
			6.4.7	Transition Process		
			6.4.8	Validation Process		
	8.5	Test Incident Reporting Process	6.4.8	Validation Process	7.5 10.5	Domain Test Reporting Application Test Reporting
			6.4.10	Maintenance Process		
					Clause 8	Asset Management in Testing
					Clause 9	Variability Management in Testing

Bibliography

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

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

[3] ISO/IEC 15940, *Systems and software engineering — Software Engineering Environment Services*

[4] ISO/IEC/TR 19759, *Software Engineering — Guide to the software engineering body of knowledge (SWEBOK)*

[5] ISO/IEC 25000, *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE*

[6] ISO/IEC 26550, *Software and systems engineering — Reference model for product line engineering and management*

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

[8] ISO/IEC 26552[1)], *Software and systems engineering — Tools and methods for product line architecture design*

[9] ISO/IEC 26553, *Software and systems engineering — Tools and methods for product line realization*

[10] ISO/IEC 26556, *Software and systems engineering — Tools and methods for product line organizational management*

[11] ISO/IEC 26557, *Software and systems engineering — Methods and tools for variability mechanisms in software and systems product line*

[12] ISO/IEC 26558, *Software and systems engineering — Methods and tools for variability modelling in software and systems product line*

[13] ISO/IEC 26559, *Software and systems engineering — Methods and tools for variability traceability in software and systems product line*

[14] ISO/IEC 26560[2)], *Software and systems engineering — Tools and methods for product line product management*

[15] ISO/IEC 26561[3)], *Software and systems engineering — Methods and tools for product line technical probe*

[16] ISO/IEC 26562[4)], *Software and systems engineering — Methods and tools for product line transition management*

[17] ISO/IEC 26563[5)], *Software and systems engineering — Methods and tools for configuration management of product line assets*

[18] ISO/IEC/IEEE 29119 (all parts), *Software and systems engineering — Software testing*

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

1) Under preparation.

2) Under preparation.

3) Under preparation.

4) Under preparation.

5) Under preparation.

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

ICS 35.080

Price based on 55 pages