**Author(s):** Käkölä, Timo

**Title:** Standards Initiatives for Software Product Line Engineering and Management within the International Organization for Standardization

**Year:** 2010

**Version:**

**Please cite the original version:**

Käkölä, T., "Standards Initiatives for Software Product Line Engineering and Management within the International Organization for Standardization," System Sciences (HICSS), 2010 43rd Hawaii International Conference on , vol., no., pp.1-10, 5-8 Jan. 2010. doi: 10.1109/HICSS.2010.348

# Standards Initiatives for Software Product Line Engineering and Management within the International Organization for Standardization

**Timo Käkölä**
University of Jyväskylä
40014 University of Jyväskylä, Finland
timokk@jyu.fi

**Abstract:** Software product line engineering is an established methodology for fast and effective development of software-intensive systems and services. To reap maximum benefits from the methodology, businesses typically need to implement *coordinated* changes in development methodologies, tools, product architectures, organizational designs, and business models. Product lines are developed in complex international software ecosystems, but there is no coordinated set of international standards for defining and leveraging the methodology. As a result, ecosystems cannot adopt standardized methods and tools for developing product lines, tool vendors face difficulties in developing tools to enable product line engineering, and universities cannot effectively set up product line engineering courses because an internationally accepted curriculum is missing. The International Organization for Standardization has initiated several projects to create a set of international standards for software product line engineering. Practitioners, researchers, and other stakeholders can contribute to these projects through their national standards bodies. This paper discusses the projects and future directions for product line standardization.

**Keywords**: International Organization for Standardization, Software product line engineering and management, Software product line body of knowledge

## 1. INTRODUCTION

To succeed in the global markets of software-intensive products, high-tech companies need to shorten the cycle time of new product development while improving product quality and service delivery and maintaining or reducing the total resources required [2;18]. This concern can be dealt through internal or external strategies. Internal strategies include global software development, where development resources are distributed globally to reap cost benefits, leverage specialized competencies, and address specific needs of geographically-defined markets [3;8;21], and software product line engineering and management, that is, the strategic acquisition, creation, and reuse of software assets [15;17;20]. External strategies include acquiring commercial off-the-shelf components and outsourcing software development, maintenance, and related services to best-in-class service providers [4;14].

This paper focuses on the software product line engineering strategy in the contexts of global software development and ecosystems. Software product line engineering is a methodology for developing software products and software-intensive systems and services faster, at lower costs, and with better quality and higher end-user satisfaction than is possible through the engineering of single systems. It has been applied and found useful in organizations worldwide. It differs from the engineering of single systems in two primary ways [20]:

1. It needs two distinct development processes: domain engineering and application engineering. Domain engineering defines and realizes the commonality and variability of the software product line, thus establishing the common software platform for developing high-quality applications rapidly within the line. Application engineering derives specific applications by strategically reusing the platform and by exploiting the variability built into the platform.
2. It needs to explicitly define and manage variability. During domain engineering, variability is introduced in all software product line assets such as domain requirements, architectural models, components, and test cases. It is exploited during application engineering to derive applications mass-customized to the needs of different customers and markets.

Organizations typically have to overcome numerous challenges to fully reap the benefits from software product line engineering. They need to implement coordinated changes in development methodologies, tools, product architectures, organizational designs, and business models [6;19]. They also need to be able to develop product lines in complex international software ecosystems. However, the software product line engineering body of knowledge is still fragmented and there is no coordinated set of international standards for defining and leveraging the software product line methodology. As a result, ecosystems cannot adopt standardized methods and tools for developing product lines, tool vendors face difficulties in developing tools to enable product line engineering, and universities cannot effectively set up product line engineering courses because an internationally accepted curriculum is missing.

The seventh subcommittee (SC 7) titled "Software and Systems Engineering" of the Joint ISO/IEC Technical Committee (JTC 1) of the International Organization for Standardization (ISO) has initiated several projects and will initiate new ones between 2010 and 2012 to create a coordinated set of international standards for


978-0-7695-3869-3/10 $26.00 

software product line engineering. The author of this paper serves as a co-editor in these projects, providing technical expertise and leadership, facilitating, and coordinating the international efforts in these projects together with the other editors. Numerous decisions need to be taken in these and future projects to define and organize the software product line body of knowledge, so it becomes even more actionable, convergent, and easier to improve by practitioners and researchers in the future.

A small group of experts within SC 7 has already conducted ample planning, research, and writing for these projects and taken some decisions to scope them appropriately. However, the projects are still in early phases. The purpose of this paper is to discuss some of the plans and issues that the projects will work on during the next few years and to solicit relevant input from practitioners and researchers to facilitate the identification and analysis of relevant content for the standards, consensus building, and decision-making. Stakeholders worldwide can contribute to the projects through their national standards bodies.

The paper is organized as follows. In Section 2, challenges involved in software product line engineering and the implications of those challenges for practitioners, researchers, educators, and other stakeholders are discussed. Section 3 discusses the background and status of software product line standards initiatives within the International Organization for Standardization. Section 4 concludes the paper and presents potential future directions for product line standardization.

## 2. CHALLENGES INVOLVED IN SOFTWARE PRODUCT LINE ENGINEERING AND MANAGEMENT

This section covers important challenges that organizations, practitioners, educators, researchers and other stakeholders face in trying to benefit from software product line engineering.

### 2.1 The need for high levels of abstraction

Software product line engineering involves higher levels of abstraction than the engineering of single-systems partly because the platforms require substantial investments, have long life cycles, and have to provide product line architectures and features generally applicable to a wide range of products, services, and markets. Without appropriate abstractions the platforms with predefined variability cannot be built and managed effectively. Highly competent individuals are required to create and manage these abstractions. On the other hand, high levels of abstraction improve productivity, for example, by enabling the automation of routine development tasks [23]. Industrially validated modeling methods and commercially available modeling tools are critically important to deal with the abstractions [13]. In addition to traditional system modeling, variability modeling is required in product line engineering to document explicitly how the applications within the product line can vary.

### 2.2 The need to cope with high levels of complexity

Software product line engineering involves higher technical, social, and managerial complexity than the engineering of single systems. Software product lines are typically larger than single systems and developed in large software ecosystems in which sets of interlinked businesses draw upon the software and hardware platforms to function as units and interact with shared international markets for software and services through the exchange of information, resources, and artifacts. Dealing with this complexity requires *well-coordinated* changes in development methodologies, processes, tools, product architectures, organizational designs, business models, and capability levels of the stakeholders involved.

Domain engineering typically requires development methodologies that can deal with large scale and long term platform development and meet stringent stability, reliability, scalability, security, and other quality requirements. Application engineering requires strong market orientation, short development cycles, and fast time-to-market. As a result, organizations wishing to deploy software product lines may not be able to choose any particular development methodology for organization-wide use, which may increase development complexity and costs [17].

Domain engineering and application engineering often need to be carried out in separate organizational units because their concerns are so different. If adequate separation is not conducted, domain engineers may end up working in short term application engineering projects, which are typically perceived to earn most or all of the financing for the business as a whole, and the long-term investments in shared platforms may be hampered. Yet, the work products of these organizations need to be successfully integrated to deliver products to markets. Without careful organizational designs, organizational complexity and costs may increase, but the organizational effectiveness may not rise accordingly.

The two engineering life cycles and enabling knowledge management systems also need to be carefully and holistically designed because the cycles are very knowledge intensive and require the management of a large number of complex dependencies between many types of artifacts. The relationships between artifacts need to be explicit and traceable to enable adequate coupling between domain and application engineering. Specifically, the relationships between domain artifacts need to be traceable within the domain engineering life cycle, the relationships between application artifacts need to be traceable within the application engineering lifecycle, and the relationships between domain and application artifacts have to be traceable across the domain and application engineering lifecycles [20]. For example, application test cases typically depend on

application requirements, but also reuse domain test cases, which depend on domain requirements [22].

Finally, the business models need to be appropriate and clear from the external and internal viewpoints. Most importantly, the organization has to identify and prioritize the market and technology domains it will be targeting and the planning horizon needs to be long enough, so it is possible to determine whether the required platform and product development and marketing investments are likely enough to produce the desired financial and other returns on investments from the targeted domains. Deep stocks of domain knowledge need to be developed about the targeted market and technology domains, so the business model must ensure direct access to the markets and technology providers. Otherwise, intermediaries may hamper or block the knowledge flows required to establish and maintain the required stocks of domain knowledge. From an internal viewpoint, the business model must facilitate the adequate (but not excessive) financing and resourcing of domain engineering. This is especially important in the early phases of building the platform(s) when substantial investments are required but their returns are not yet materializing [17].

### 2.3 The fragmented body of knowledge

While many businesses have been able to increase their organizational efficiency and effectiveness through software product line engineering, the extant software product line body of knowledge remains fragmented, making it challenging for educators to teach and for students and practitioners to understand and apply software product line concepts. An internationally accepted academic educational curriculum and readily available teaching materials for software product line engineering are missing, hampering the effective set up and execution of product line engineering courses in universities and other institutes of higher education. The only well known curriculum has been established by the Software Engineering Institute, but it is proprietary and targeted to practitioners. It thus cannot be applied directly in universities.

Most advances in the field have been presented in individual scientific papers but, in practice, few, if any, of the methods and tools presented in the papers are so widely used in the industry globally that they could be considered as de-facto standards for software product line engineering. International standards do not cover software product line engineering either. Individual papers cannot have both the holistic scope and the details required to help businesses take coordinated product line adoption actions to redesign their business models, processes, and structures, knowledge management systems, and product line architectures appropriately.

Books can better meet the scope requirements than individual research papers but there are only a few books that take a scientific and holistic enough view to enable software product line engineering education in universities. Pohl, Böckle, and Van der Linden [20] completed in 2005 the first and, so far, the only book about software product line engineering explicitly targeted to students in undergraduate and graduate level university courses. It covers all the practices and concepts that need to be mastered to implement product lines. However, it has been written on a relatively high level of abstraction and the industrial validation of the proposed methods and tools is in early phases, making the practical application of the methods and tools challenging. The book of Clements and Northrop [5] is another important component of the product line engineering body of knowledge. It primarily supports the practitioner-oriented curriculum of the Software Engineering Institute. Due to their different focuses, the two books might complement each other well, when properly integrated under a common curriculum. However, such an integrated curriculum is nontrivial to develop, partly because Pohl et al. [20] and Clements and Northrop [5] reflect, respectively, European and North American product line research traditions relying on partly different concepts, terminologies, and methods to describe, develop, and deploy product lines. As a result, most students in software engineering, management information systems, information technology, marketing, and other relevant disciplines cannot develop holistic, inter-disciplinary views about software product line engineering and its organizational, technological, business, social, psychological, and other implications during their studies.

### 2.4 The lack of tools

The fragmented body of knowledge and the lack of international software product line standards also hurt commercial software tool vendors. Interoperable and even integrated information systems are critical to support knowledge management throughout the domain and application engineering life-cycles [16] and during variability modeling and resolution [1]. Yet, commercially available and industrially validated software tools to implement such systems for product line engineering are scarcely available partly because the markets do not know what to expect from such tools and thus remain small and fragmented. Tool vendors have typically circumvented the problem (1) by focusing on features that support the engineering of single systems well through standardized general purpose languages and methods and (2) by enabling extension mechanisms that let organizations tailor their own methods and tools for purposes such as variability modeling.

For example, to model the variability of product lines, two approaches have been proposed in the literature: integrated and orthogonal variability modeling. The first, traditional approach has been to integrate variability modeling in the systems modeling notation, typically Unified Modeling Language™ (UML) [7;10], by appropriately extending the metamodel of the notation through profiles. Extended UML metamodels enabling both system and variability modeling are available in the literature [1] and most general purpose

UML modeling tools support the design and use of the extended metamodels through profiles [24;25]. In this approach, variability models can only be edited by people who use UML tools with the same extended UML metamodel. An organization can thus implement variability modeling practices by adopting a variability metamodel available in the literature; adapting the metamodel to its needs as necessary; establishing organization-wide policies, incentive schemes, and training mechanisms to enforce variability modeling using the metamodel; and codifying the metamodel into the modeling tool(s) used organization-wide.

However, a thorough literature review conducted for this research paper has revealed no literature about such variability modeling implementations in organizations. Indeed, agreeing upon and institutionalizing a new extended UML meta-model in any large organization is likely to be very challenging considering that UML is complex in itself to learn and use and extensions will increase the complexity. Moreover, spreading the variability information across different system models would make it very difficult to keep the information consistent and unambiguous and to provide stakeholders with holistic views of software variability [20, p. 74-75]. Even if the meta-model was agreed upon and instituted organizationally, it would not be enough because software product line engineering typically involves numerous partners and other external organizations in complex global ecosystems. To establish variability modeling practices using UML tools in such ecosystems, all the organizations involved would have to agree upon a common extended UML meta-model. Based on the available literature and the author's own experiences in such global ecosystems, it will be unlikely that variability modeling practices based on such metamodel extensions would become common. A more feasible option may be to officially extend the ISO/IEC 19501 (UML) standard [10] to cover variability modeling. Such an extension is beyond the scope of this paper and the ISO is not currently pursuing it.

The second variability modeling approach, orthogonal variability modeling, distinguishes between a variability model and a system model [20]. An orthogonal variability model represents a cross-sectional, holistic view of variability across all associated software development artifacts such as requirements, designs, software implementations, and test cases. Models with the highest levels of abstraction are used to represent external variability from the viewpoint of market segments and customers, whereas the lower abstraction levels manifest internal (mostly technical) variability from the viewpoint of the product line provider. Pohl et al. [20, Ch. 4] present a comprehensive metamodel for orthogonal variability modeling that enables product line engineers to capture variation points, that is, the items that vary; variants defining how the variable items can vary; and constraints between variants, between variants and variation points, and between variation points.

Orthogonal variability models are easier to apply in practice than integrated models partly because organizations can continue to use the systems modeling notations such as UML they are familiar with. No changes in the metamodels of systems modeling notations are needed because orthogonal variability models are created separately and associated with the system models through traceability links. Orthogonal variability models also scale better than integrated ones. They usually describe the variability using graphical notations.

However, orthogonal variability modeling is not yet used in the industry partly because there are no commercially available modeling tools to support it. The best way to improve tool support may be the international standardization of the metamodel for orthogonal variability modeling. Vendors would then have a baseline for developing their tools and markets would know better what benefits and services to expect from variability modeling and the supporting tools. The metamodel presented by Pohl et al [20, Ch. 4] can be used as one basis of standard development because it captures variation points, variants, and the other meta-classes and their relationships deemed most important in the variability modeling literature.

### 2.5 Summary

Software product line engineering research and practice has produced a very comprehensive body of knowledge that has already been used successfully in numerous organizations. However, over time the body of knowledge has become divergent, consisting of a myriad of concepts, models, methods, and tools that are partially overlapping, inconsistent, competing, and possibly even conflicting.

During the next decade, the field must focus more on consolidating its body of knowledge. Indeed, the field has now matured to a stage where the next level of development requires coordinated actions beyond the research community. Markets naturally have a key role in determining which research deliverables and other stocks of knowledge produce most value in practice, thus facilitating the consolidation of the software product line body of knowledge. International standardization efforts are also needed to determine those parts of the body of knowledge that are stable and coherent enough for standardization purposes.

## 3. STANDARDIZATION INITIATIVES

To address many of the diverse challenges software product line researchers and practitioners face (Section 2), a set of interrelated software product line engineering standards will be created by ISO/JTC1/SC7. This section describes the background for the software product line standardization initiative and presents and analyzes the preliminary reference model that is being developed to guide more specific standardization projects in the future.

### 3.1 Background

ISO/JTC1/SC7 decided to initiate activities related to software product line engineering standardization in its plenary meeting in Helsinki in May 2005. It appointed the author of this paper and professor Dan Lee (from Korea Advanced Institute of Technology) to lead an international group of experts in a project studying the need for new and/or revised standards in the field of requirements engineering. At that time, the author was involved with the largest software engineering research project series (ESAPS, CAFÉ, and FAMILIES) conducted in Europe by that time. The series of projects focused on product line engineering and FAMILIES, the concluding project, was being completed. Substantial advances in the tools and methods related to software product line requirements engineering had been created in the project series [15;17;20]. No international standards existed for such methods and tools because the existing standards focused on requirements engineering of single systems. In response to the appointment, the author and Dan Lee reviewed requirements engineering standards and proposed that ISO/JTC1/SC7, among other issues, should create a new standard for software product line requirements engineering. The proposal was accepted in May 2006 and a new project ISO/IEC 29118 was started.

During the next two years, a small team including Dan Lee and the author created several ISO/IEC 29118 standard drafts. During the process, it became evident that the scope of the 29118 project was too wide. Software product line requirements engineering involves parts of both domain and application engineering lifecycles. Because there is no standard describing the lifecycles and the central concepts of software product line engineering, a reference model for the two lifecycle processes and a thorough glossary were incorporated in the 29118 drafts. Some drafts were more than 100 pages long with several appendixes.

ISO/JTC1 policies have recently started to emphasize the creation of sets of small, interrelated, and coordinated standards. For each set, one standard typically outlines the overall architecture, that is, the concepts to be probed in individual standards within the set and the relationships between them. In broad, complex, and rapidly developing fields such as software product line engineering, the standard creation process is typically ongoing, that is, once an international standard is accepted and released, its revision starts immediately so a new release replacing the previous one (and possibly some other standards) will be ready about five years later. It is easier to create, agree upon, and maintain small and well-scoped standards than a few large ones. The drawbacks with the approach relate mainly to keeping numerous interrelated standards within each set consistent and to the readability and understandability of each set when there is no one document covering the entire scope of the standard set holistically and in depth.

SC 7 has decided to create a set of interrelated software product line engineering standards. The 29118 project has been terminated in 2009 and its contents have been divided into two new projects. ISO/IEC 26520 [11] will establish a reference model for software product line engineering, covering phases of the domain and application engineering life-cycles on a high level of abstraction. ISO/IEC 26521 [12] details the domain and application requirements engineering processes. During the next decade, new projects will create three standards for the other commonly accepted life-cycle phases: product line architecting, realization, and testing. There will also be standards for technical management (ISO 26525) and organizational management (the standard number has not yet been assigned) practices.

The set of standards is intended to benefit businesses in various industries, tool vendors, researchers, and research institutes investigating, facilitating, and/or leveraging software product line engineering practices.

### 3.2 Review of ISO/IEC 26520 Draft from May 2009

The draft of the ISO/IEC 26520 is intended to**:**

- Enable its users to holistically understand, adopt, and enact the domain and application engineering lifecycles.
- Enable its users to evaluate and select relevant methods and tools based on business and user-related criteria.
- Help development teams specify, verify, and validate engineering and management practices for existing or envisioned product lines based on the product line practices described in the standard.
- Provide a reference model for software product line engineering and management, that is, an abstract representation of the domain and application engineering lifecycles and enabling core asset management, organizational management, and technical management processes that will be detailed in the other software product line standards.
- Help tool vendors develop interoperable tool suites and features supporting the domain and application engineering life-cycles and their phases and communicate about their tools to the markets.

In the preliminary reference model (Figure 1), software product line engineering consists of domain engineering and application engineering life-cycles. The phases of the two life-cycles have been aligned with the framework of Pohl et al. [20, p. 22], so the users of the envisioned standard can find details about the life-cycles and their phases from one well-respected source. The life-cycles should be loosely coupled, synchronized by platform releases, and enacted based on different life-cycle models as they are typically performed with different quality criteria and objectives in mind. In addition, the reference model emphasizes core asset management, organizational management, and technical management processes.

**Domain engineering**

Domain engineering consists of the following phases: product line scoping, domain requirements engineering, domain design, domain realization, and domain testing (c.f., [20, p. 22]). Product management is responsible for *product line scoping* (or, synonymously, product line roadmapping or product portfolio management) that

- defines the products that will constitute the product line and the common and variable features that are visible to markets,
- analyzes the products from an economic viewpoint, and
- schedules and monitors the development and marketing of the product line and its products.

Product line scoping is divided into three subprocesses (c.f., [17, p. 31]):

- *Product portfolio scoping* determines the product roadmap for the products the product line organization should be developing, producing, marketing, and selling; the common and variable features that should be provided to reach the long and short term business objectives of the product line organization; and the schedule for introducing products to markets.
- *Domain scoping* draws upon the product types to identify the functional areas most important to the envisioned product line. The areas must provide sufficient reuse potential to justify the product line.
- *Asset scoping* is used to identify the core assets (i.e., domain artifacts the common and variable features of the product line can reuse) and estimate their costs and benefits in order to determine whether an organization should launch a product line.

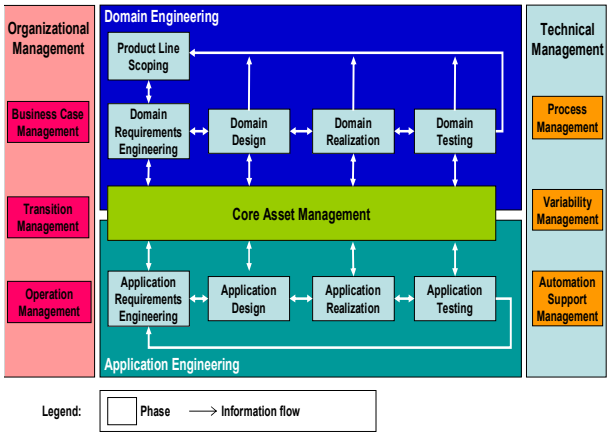


**Figure 1. The preliminary reference model of the ISO/IEC 26520 draft for software product line engineering and management [11].**

*Domain requirements engineering* refines the common and variable high-level features for the products identified during *product line scoping*; constructs a detailed enough requirements specification, including the orthogonal variability model, to guide *domain design, realization, and testing*; and provides feedback to product management with respect to the changes required in the feature sets and the product roadmap as a whole. It consists of *domain requirements elicitation, analysis, specification, and validation* phases. *Domain requirements elicitation* identifies product line stakeholders as broadly as possible and captures the common and variable domain requirements the stakeholders can foresee over the lifetime of the product line. *Domain requirements analysis* identifies and refines the elicited domain requirements. Thorough negotiations with the stakeholders are typically necessary to ensure there are enough common and variable features to justify product lines from the economic viewpoint. *Domain requirements specification* documents the common and variable requirements. Specifications may include symbolic placeholders, where application requirements engineering can incorporate product-specific requirements. During *domain requirements validation* the stakeholders verify and validate that that the right domain requirements have been specified correctly. Domain requirements are then baselined. During *domain requirements management* they can only be changed through systematic change management and impact analysis policies governing how changes in the domain requirements are proposed, reviewed, and accepted. Domain requirements are domain artifacts managed by domain requirements engineering.

*Domain design* develops a *product line reference architecture* enabling the realization of the common and variable requirements, evaluates the architecture from the viewpoints of functional and quality requirements, and manages changes related to the architecture. The reference architecture defines a generic structure and a set of rules all applications within the product line must follow to successfully reuse the common features and some of the variable features. It also incorporates new internal variable features to facilitate the technical implementation of various product variants during application engineering. It must have a long life cycle and accommodate the needs of all applications reasonably well, so it should be periodically evaluated and improved as necessary. To facilitate evaluation, change management, and impact analysis, the architecture has to fully traceable to the respective domain requirements. *Domain reference architectures* are *domain artifacts* managed by *domain design*.

*Domain realization* buys, designs, and implements the common and variable *realization artifacts*, including the reusable components, interfaces, and the supporting infrastructure, and partially validates them with respect to the common and variable requirements and the structures and rules of the *reference architecture*. The outcomes of *domain realization* are loosely coupled, configurable, and reusable software components and interfaces that implement the common and variable features afforded by the *software platform*. *Domain realization* is also responsible for managing changes related to these interfaces and components. It does not build executable applications. As a result, validation and change management depend on full traceability (1)

between domain realization artifacts and application realization artifacts that together constitute executable applications and (2) between domain realization artifacts and the other domain artifacts. For example, it must be known which versions of which domain components and interfaces are used in which applications. *Domain components* and *interfaces* are *domain artifacts* managed by *domain realization*.

*Domain testing* creates and validates *domain test artifacts*; uses the artifacts to validate the *domain requirements*, the *product line reference architecture*, and the *domain realizations* created in the previous phases of the *domain engineering life-cycle*; and manages changes related to the *domain test artifacts*. The domain test artifacts must also be reused in application testing to test the common features and those variable features that have been bound for the application. Domain testing in the ISO/IEC 26520 draft refers to the review, validation, and verification of domain artifacts and to the testing of executable software implementations. The domain testing of implementations involves the creation of domain test cases for the common and variable features realized by the platform based on the inputs from domain requirements engineering, domain design, and domain realization; the application of the test cases to executable implementations; the observation of the outcomes; and the detection and correction of defects. Domain test artifacts such as domain test cases typically have to be created and validated in close collaboration with application testing, when executable applications are available. *Domain test artifacts* are *domain artifacts* managed by *domain testing*.

**Application engineering**

Application engineering develops individual systems on top of the platform established in domain engineering. It deals with less complexity and shorter development times than domain engineering because large parts of engineering have been moved to domain engineering. It is directly involved with customers and thus needs to deal with rapidly changing market needs effectively while using the platform offerings to maximum possible extent.

In the preliminary ISO/IEC 26520 reference model, application engineering follows the life-cycle described by Pohl et al. [20, p. 22]: application requirements engineering, application design, application realization, and application testing. Therefore, these life-cycle phases are only described briefly in the following subsections.

*Application requirements engineering* identifies the specific requirements for individual products. It starts from existing common and variable requirements to maximally leverage the product line platform. It also has an important role in providing insights to domain requirements engineering in order to guide platform development especially in the early phases of the product line creation.

*Application design* derives an instance of the product line reference architecture and adapts it, so the resulting application architecture conforms to the application requirements. The application architecture should be consistent with the reference architecture to enable the reuse of domain artifacts.

*Application realization* implements products by drawing upon the application requirements and architecture; reusing, configuring, and adapting existing domain components and interfaces; and building new components and interfaces to enable application-specific functionality.

*Application testing* validates the final applications against the application requirements by drawing upon domain test artifacts to test common and variable features derived from the platform; creating new application test artifacts for application-specific features; performing application-specific tests; and analyzing the results and taking corrective actions.

**Organizational management**

*Organizational management* refers to organizational-level ongoing practices that are necessary for the successful introduction and institutionalization of the domain and application engineering life-cycles in organizations (c.f., [5;19]). Organizations cannot reap maximum benefits from software product line engineering without solid business cases, long-term commitments of top executives, appropriate organizational designs, effective transitioning of people from the engineering of single systems to engineering product lines, and consistent orchestrations of daily product line routines. The ISO/IEC 26520 draft outlines three organizational management practice areas: business case management, transition management, and operations management.

Business cases in the context of software product lines serve two purposes [19]: (1) justifying the effort to adopt the product line approach for building products and (2) deciding whether or not to include a particular product as a member of a product line. This practice area is challenging from the viewpoint of the structure of the ISO/IEC 26520 draft because the same practices are also incorporated in the domain scoping phase of the preliminary reference model, creating redundancy within the model. However, it may be justified to keep this practice area in *organizational management* because the building of business cases is a complex, multi-disciplinary, and strategic ongoing activity at least in large organizations that run multiple product lines. Competencies for building business cases in such organizations must be shared across all product lines and products and thus be located in the organizational level. Moreover, organization-wide data collection and measurement systems need to be instituted to track on an ongoing basis how well product line organizations are meeting the goals specified in the business cases. Such issues are not addressed by the domain and application engineering life-cycle models specified in the draft.

*Transition management* practice area incorporates all practice areas that according to Northrop and Clements [19] are needed to transition an organization from single

systems engineering to product line engineering: funding, launching and institutionalizing, structuring the organization, training, and organizational planning and risk management. Launching a software product line requires the creation of an adoption plan to describe how the two life-cycles and the supporting organizational and technical management practices will be appropriately rolled out across the organization. The product line strategy can be considered institutionalized within organizations when the organizations have extensive core asset bases and supporting organizational structures, processes, information systems, and business models in place and when their developer communities routinely use the domain artifacts to develop products.

*Operations management* practice area [19] describes (1) the organizational units and stakeholders involved, (2) the interconnections among organizational units carrying out domain engineering, application engineering, and management, and (3) how the organizational units

- produce and evolve domain artifacts
- define and evolve the production plan
- use the domain artifacts and production plan to field products and
- monitor and improve the health and profitability of the product line.

It should be noted that the planning of the specific standard for organizational management practice areas is in early phases. There is little published research and data available globally from software product line organizations related to the relative importance of the practice areas. It is thus challenging to determine which particular areas are important enough be incorporated in the standard. When this work progresses and more evidence is obtained, other organizational management practice areas are likely to be incorporated in the ISO/IEC 26520 reference model. For example, acquisition management is a key practice area because both application and domain engineering units may outsource or offshore large chunks of engineering.

It should also be noted that the organizational management practice areas in the draft are descriptive because they follow mainly the lessons from the software product line practice initiative of the Software Engineering Institute [19]. The draft thus does not provide a staged representation similar to the staged representation of the CMMI. A staged representation would provide organizations with a roadmap through which they could raise the maturity of software product line practices by following a proven sequence of improvements, beginning with basic practices and tools and progressing through a predefined path of successive levels, each serving as a foundation for the next.

Van der Linden et al. [17] present the Family Evaluation Framework, providing organizations with such a roadmap. The framework can be used to measure organizations' product line abilities from the viewpoints of business, domain and application engineering life-cycles, product line reference architecture, and organization design. Improvements can then be focused on areas where most value can be created. The framework can thus be much more useful than simple descriptions. However, it has not yet been extensively validated in practice. The future development of the organizational management standard thus calls for thorough empirical validation of the framework. If the framework is found valid and useful, it can be seriously considered for inclusion in the reference model and the specific organizational management standard.

**Technical management**

Technical management practices are management practices necessary for the development and evolution of both domain artifacts and applications [19]. In the draft, the *process management* practice area includes all practice areas (except for the *scoping* practice area that is already incorporated in *product line scoping* and *business case management*) defined by Northrop and Clements [19] without significant changes: configuration management, make/buy/mine/commission analysis, measurement and tracking, process discipline, technical planning, technical risk management, and tool support. They are not elaborated in this paper.

*Variability management* practice area helps domain and application engineers maintain the orthogonal variability model of a product line through five practices: (1) variability modeling, (2) traceability management, (3) variability annotation, (4) variability validation, and (5) variability control.

*Variability modeling* supports domain and application engineers in developing the variability models in appropriate levels of detail and through consistent notations.

*Traceability management* helps domain and application engineers establish and maintain links between the variability model and associated domain and application artifacts. The costs and benefits of traceability links should be analyzed to determine appropriate level of traceability. Having too much or too little traceability is costly [16].

*Variability annotation* helps domain and application engineers add descriptions to variability models and their elements, documenting, for example, whether a variation point deals with external or internal variability.

*Variability validation* helps domain and application engineers and other stakeholders review or inspect the orthogonal variability model and its documentation and associations with domain and application artifacts.

*Variability control* helps domain and application engineers manage changes related to variation points, variants, variability constraints, and traceability links in the orthogonal variability model.

In the draft, *variability management* practice area belongs to *technical management*. However, the orthogonal variability model is usually the most important single domain artifact within a product line. Therefore, it may be more logical to incorporate *variability management* in the *core asset management* practice area in future drafts of ISO/IES 26520.

**Core asset management**

*Core asset management* practice area involves a set of information systems and associated services for storing, mining, and managing changes in the domain artifacts resulting from domain engineering and the relationships (e.g., traceability links) between them. *Core asset management* represents the most important difference between the preliminary reference model and the framework of Pohl et al. [20]. Pohl et al. [20, p.22] emphasize the roles of domain and application artifacts, including variability models, by visually representing domain artifacts as outputs of respective domain engineering phases that can be stored in domain artifact repositories and used as inputs to the respective application engineering phases. For example, domain requirements engineering produces domain requirements, which are reused or adapted by application requirements engineering. Pohl et al. [20] also visually represent application artifacts as application-specific outputs of the respective application engineering phases that can be stored in application-specific repositories. For example, when adapted domain requirements and new application-specific requirements together with the variability model are stored in the application-specific requirements repositories, it is easier for domain requirements engineers to analyze the application-specific requirements later. Based on the analysis, domain requirements engineers can then determine whether some of the requirements should be generalized as domain requirements and stored in domain requirements repositories for reuse in other applications.

In the preliminary reference model of ISO/IES 26520, domain and application artifact repositories and their roles in sharing knowledge between domain and application engineering are not visible. Instead, *core asset management* implicitly incorporates the repositories. However, in the draft standard *core asset management* includes comprehensive change and configuration management processes, which are already incorporated in the domain engineering life-cycle. On the other hand, core asset management in the draft standard does not include application artifact repositories at all.

The coverage of *core asset management* needs to be revised extensively in the future 26520 drafts to remove redundancy and fix the omissions. It is probably best to keep the change management of various domain artifacts in the respective domain engineering phases and only provide the domain and application artifact repositories and associated information management services through core asset management.

## 4. CONCLUSIONS

This paper has identified and justified the need for international standards in the area of software product line engineering and management. It has outlined the strategy ISO/JTC1/SC7 is taking to bring about a set of software product line engineering and management related standards. It has also presented and analyzed the preliminary reference model for software product line engineering and management to which all subsequent software product line standards will be linked. The set of standards is meant to advance (1) software product line practices in organizations in various industries and (2) the convergence and further accumulation of the software product line body of knowledge.

The set is envisioned to be fairly well aligned with the books of Pohl et al. [20] and Van der Linden et al. [17]. It thus relies on orthogonal variability modeling that, so far, has not been extensively validated in various industries. This implies that design science research [9] is needed to develop industrially feasible tools for orthogonal variability modeling and that behavioral science research (e.g., case studies and action research) is needed to validate the orthogonal variability modeling approach in practice.

Most of the tools and methods available in the extant software product line body of knowledge have not yet been fully validated in various industries. Therefore, this paper and the entire software product line standardization initiative of the ISO call for more extensive collaborative efforts between researchers and practitioners to further improve the relevance and validity of the available tools and methods. For example, the Family Evaluation Framework [17] can provide organizations with roadmaps through which they can choose the components of the product line body of knowledge most appropriate to their needs, accelerating the adoption of product line practices. When the framework has been validated through empirical research and appropriately revised, it may be incorporated in an international standard during the next decade.

To facilitate industrial applications of software product line engineering, other new standardization projects, that have not yet been planned officially, will also be needed. For example, the variability metamodel for orthogonal variability modeling is not yet a part of any standardization project in ISO. Establishing a new standard solely for the variability metamodel may be the best option because it enables the evolution of the metamodel independently from other standards. The metamodel can also be expected to remain quite stable because the extant literature has already converged quite well concerning the metaclasses (e.g., variation point and variant) of the metamodel.

## 5. REFERENCES

1. Bayer, J., Gerard, S., Haugen, Ø., Mansell, J. X, Møller-Pedersen, B., Oldevik, J., Tessier P., Thibault, J-P and Widen T. (2006). Consolidated Product Line Variability Modeling. In T. Käkölä & J.C. Dueñas (Eds.), Software Product Lines: Research Issues in Engineering and Management, Springer, 195-241.

2. Brown, S. L. and Eisenhardt, K.M. (1995). Product Development: Past Research, Present Findings, and Future Directions. Academy of Management Review 20(2), 343-378.

3. Carmel, E. and Agarwal, R. (2001). Tactical Approaches for Alleviating Distance in Global Software Development. IEEE Software, 18(2), 22-29.

4. Carmel, E. and Agarwal, R. (2002). The Maturation of Offshore Sourcing of Information Technology Work. MIS Quarterly Executive, 1(2), 65-78.
5. Clements, P. and Northrop, L. (2001). Software Product Lines: Practices and Patterns. Addison-Wesley: SEI Series in Software Engineering.
6. Cusumano, M. A., and Selby, R. W. (1995). Microsoft® Secrets. New York, NY: Free Press.
7. Gomaa, H. (2004). Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Software Product Lines, Third International Conference, SPLC 2004.
8. Herbsleb, J.D. and Moitra, D. (2001). Guest Editors' Introduction: Global Software Development. IEEE Software, 18(2), 16-20.
9. Hevner, A.R., March, S.T., Park, J. and Ram, S. (2004). Design Science in Information Systems Research. MIS Quarterly, 28(1), 75-105.
10. ISO (2005). ISO/IEC 19501 Information Technology-Open Distributed Processing-Unified Modeling Language (UML) Version 1.4.2. International Organization for Standardization, http://www.iso.org.
11. ISO (2009a). ISO/IEC NP 26520 Software and Systems Engineering – Reference Model for Software and Systems Product Lines. ISO/IEC JTC1/SC7 WG4 Working draft 1.0.
12. ISO (2009b). ISO/IEC NP 26521 Software and Systems Engineering – Tools and Methods of Requirements Engineering and Management for Software and Systems Product Lines. ISO/IEC JTC1/SC7 WG4 Working draft 1.0.
13. Koivulahti-Ojala, M. and Käkölä, T. (2010). Framework for Evaluating the Version Management Capabilities of a Class of UML Modeling Tools from the Viewpoint of Multi-site, Multi-partner Product Line Organizations. In Proceedings of 43rd Hawaii International Conference on Systems Sciences (HICSS-43). IEEE.
14. Käkölä, T. (2008). Best Practices for International eSourcing of Software Products and Services. In Proceedings of 41st Hawaii International Conference on Systems Sciences (HICSS-41). IEEE.
15. Käkölä, T. and Dueñas, J. (Eds.) (2006). Software Product Lines: Research Issues in Engineering and Management. Springer.
16. Käkölä, T., Koivulahti-Ojala, M., and Liimatainen, J. (2009). An Information Systems Design Product Theory for the Class of Integrated Requirements and Release Management Systems. Software Process: Improvement and Practice (in press).
17. Van der Linden, F., Schmid, K. and Rommes, E. (2007). Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer.
18. Meyer, M.H. and Selinger, R. (1998). Product Platforms in Software Development. Sloan Management Review, 40(1), 61-74.
19. Northrop, L. and Clements, P. (2007). A Framework for Software Product Line Practice, Version 5.0. Software Engineering Institute. http://www.sei.cmu.edu/productlines/framework.html.
20. Pohl, K., Böckle, G., and Van der Linden, F. (2005). Software Product Line Engineering: Foundations, Principles, and Techniques. Springer.
21. Ramasubbu, N., Krishnan, M. S. and Kompalli, P. (2005). Leveraging Global Resources: A Process Maturity Framework for Managing Distributed Development. IEEE Software, 22(3), 80-86.
22. Reuys, A., Reis, S., Kamsties, E., and Pohl, K. (2006). The ScenTED Method for Testing Software Product Lines. In T. Käkölä & J.C. Dueñas (Eds.), Software Product Lines: Research Issues in Engineering and Management, Springer, 479-520.
23. Rosenmuller, M., Siegmund, N., Saake, G. and Apel S. (2008). Code Generation to Support Static and Dynamic Composition of Software Product Lines. In Proceedings of the 7th International Conference on Generative Programming and Component Engineering, 3-12.
24. Ziadi, T., Hélouët, L. and Jézéquel, J.M. (2003). Towards a UML Profile for Software Product Lines. In Proceedings of the 5th International Workshop on Product Family Engineering (PFE-5). Lecture Notes in Computer Science, vol 3014, Springer, 129–139
25. Ziadi, T. and Jézéquel, J.-M. (2006). Software Product Line Engineering with the UML: Deriving Products. In T. Käkölä & J.C. Dueñas (Eds.), Software Product Lines: Research Issues in Engineering and Management, Springer, 556-588.