Fraunhofer Institut
Experimentelles
Software Engineering

Principles of Software Product Lines and Process Variants

Authors:
Joachim Bayer
Stefan Kettemann
Dirk Muthig

PESOA
Process Family Engineering in Service-Oriented Applications

BMBF-Project

IESE-Report No. 122.06/E
Version 1.0
February 28, 2004

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competitive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach (Executive Director)
Prof. Dr. Peter Liggesmeyer (Director)
Fraunhofer-Platz 1
67663 Kaiserslautern

PESOA-Report No. 03/2004

PESOA is a cooperative project supported by the federal ministry of education and research (BMBF). Its aim is the design and prototypical implementation of a process family engineering platform and its application in the areas of e-business and telematics.
The project partners are:

- DaimlerChrysler Inc.
- Delta Software Technology Ltd.
- Fraunhofer IESE
- Hasso-Plattner-Institute
- Intershop Communications Inc.
- University of Leipzig

PESOA is coordinated by
Prof. Dr. Mathias Weske
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam

www.pesoa.org

Abstract

Emerging technologies and new trends such as web services and the increasing collaboration between companies have reinforced the importance of the business process for the design and development of future software. This and the increasing need for individualization via mass customisation requires technologies that facilitate the efficient customizing of processes – or in other words, the efficient management of process variants. Today, the leading approach for successful software mass customisation is software product line technology. It therefore represents a key technology for managing process variants.
This report presents the main principles of software product line technology and sketches their application for the management of process variants.

Keywords: PESOA, Software Product Lines, Business Process Modeling, Software Variants.

Copyright © Fraunhofer IESE 2006

Table of Contents

1	**Introduction**	**1**
1.1	Project Context	2
1.1.1	Fraunhofer IESE	2
1.1.2	PuLSE	3
1.1.3	Goal	4
1.2	Outline	5
2	**Principles of Software Product Lines**	**6**
2.1	Product Line Engineering	6
2.2	Product Line Concepts	12
2.2.1	Commonality	12
2.2.2	Variability	12
2.3	Product Line Infrastructures	13
2.3.1	Product Line Information	13
2.3.2	Elements of Product Line Infrastructures	17
2.3.3	Processes as Variability Driver	22
3	**Process Variants**	**23**
3.1	Origins and System Evolution	23
3.2	Process Modeling – Standards	25
3.3	Need for Process Variants	26
3.4	Modeling Process Variants	27
3.5	An Infrastructure for Managing Process Variants	30
4	**Conclusion and Outlook**	**32**
List of Abbreviations		**34**
References		**35**

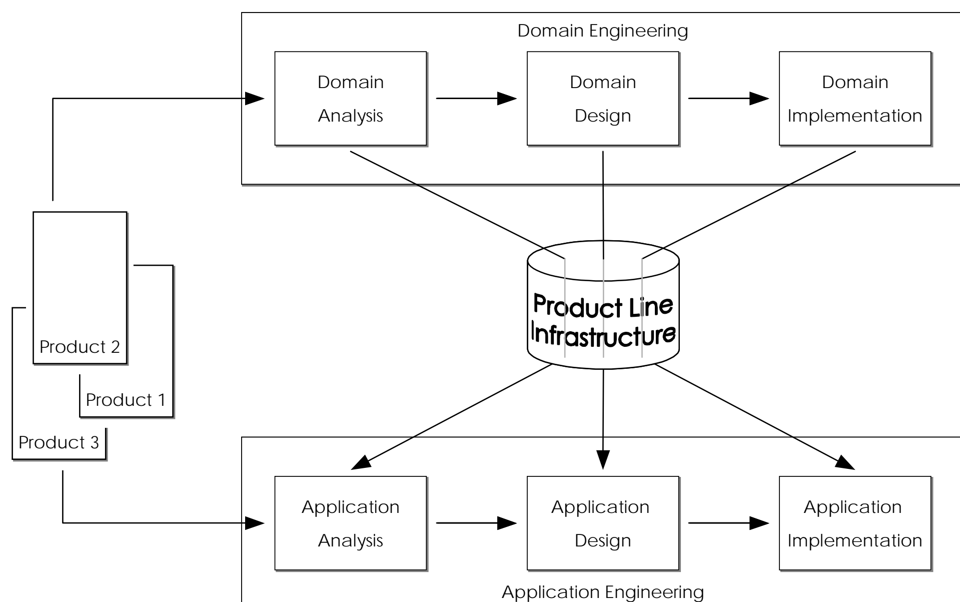
Copyright © Fraunhofer IESE 2006

1 Introduction

A software product line is a set of similar software systems that are developed and maintained together [Don00] [Cha02]. The basic idea that underlies product line engineering is to exploit the similarities of different systems and to reuse common parts of them. A product line has been defined as “a family of products designed to take advantage of their common aspects and predicted variability” [WL99]. Another often used definition was proposed by the Software Engineering Institute (SEI): “A software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.” [CN02].

Figure 1 shows the generic product line engineering life cycle that is split into the two phases domain engineering and application engineering. Domain engineering itself is decomposed into domain analysis (i.e., the scoping of the domain and the creation of reusable requirements that encompass the domain), domain design (i.e., the creation of a common architecture for all systems in the domain), and domain implementation (i.e., the implementation of reusable assets used to build the systems in the domain).

Figure 1. Generic Product Line Engineering Life Cycle.



Copyright © Fraunhofer IESE 2006

The result of domain engineering is a product line infrastructure that contains assets that are used during application engineering. Application engineering is also split in three phases (application analysis, application design, and application implementation), in which the assets that have been created during domain engineering are used to build actual systems in the domain.

There are numerous approaches proposed in literature for software product line engineering [Cha02, Don00]. However, it has not been used yet from a process perspective.

1.1 Project Context

PESOA is a cooperative project financed by the federal ministry of education and research (BMBF). Its aim is the design and prototypical implementation of a process family engineering platform and its application in the areas of e-business and telematics. This will be achieved by enhancing the approved technologies from the area of domain engineering, product line engineering and software generation with new methods from the area of workflow management.

Fraunhofer IESE is internationally recognized as one of the leading institutes in product line technology. For example, Fraunhofer IESE developed PuLSE™ (Product Line Software Engineering) - a product line method for enabling the conception and deployment of software product lines in a large variety of enterprise contexts (see section 1.1.2).

In the context of PESOA, the IESE will enhance process methodology with innovative product line technology. Vice versa, approved product line methods – in particular PuLSE – will be extended with the workflow perspective.

1.1.1 Fraunhofer IESE

The Fraunhofer Institute for Experimental Software Engineering (IESE) focuses on applied research, development and technology transfer in the areas of innovative software development approaches, quality and process engineering, product lines, as well as continuous improvement and organizational learning. To prepare industrial software developers and users for current and future information technology challenges, new techniques, methods, processes, and tools are being developed to base software development on sound engineering principles. The IESE thus provides competence, as well as the methods and tools necessary to mature industrial software development practices and give companies a competitive market advantage.

Copyright © Fraunhofer IESE 2006

Fraunhofer IESE offers an extensive portfolio of services to companies of any size or type. Partner companies work in areas where the quality of the products and services depends heavily on the quality of the supporting software - like automobile production, telecommunications, transportation, trade, banking and insurance, software production. Services range from case-oriented consulting to setting up new structures and processes in software development (Learning Software Organization). They include support for introducing continuous improvement programs as well as selecting, adapting, evaluating, and introducing innovative software development approaches. Special attention is given to the growing lack of software experts. By offering job-oriented education programs as well as technology related education and training courses, the Fraunhofer IESE helps create new chances for people entering the field of software engineering from a different professional background. It also helps secure jobs and reduce the problems companies have with existing personnel lacking up-to-date education in Software Engineering.

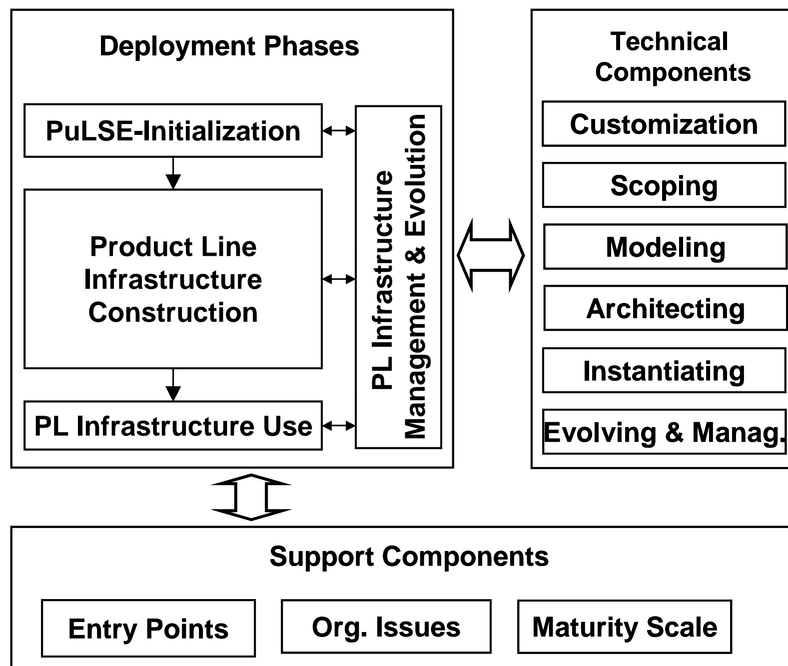
1.1.2 PuLSE

Product line software engineering aims at creating generic software assets that are reusable across a family of target products. PuLSE™ (Product Line Software Engineering) is a method for enabling the conception and deployment of software product lines in a large variety of enterprise contexts [BFK+99].

The components of the PuLSE method are shown in Figure 2. The life cycle of a software product line in PuLSE is split into the following deployment phases: initialisation, product line infrastructure construction, usage, and evolution. PuLSE provides technical components for the different deployment phases that contain the technical know how needed to operationalise the product line development. The technical components provided by PuLSE are customisation, scoping, modelling, architecting, instantiating, as well as evolving and managing. The technical components are customisable to the respective context. Customisation of PuLSE to the context where it will be applied ensures that the process and products are appropriate.

Copyright © Fraunhofer IESE 2006

Figure 2. Pulse Method.



In the initialisation phase, the other phases and the technical components are tailored. Through this tailoring of the technical components, a customized version of the construction, usage, and evolution phases of PuLSE is created.

The principle dimensions of customisation are the nature of the application domain, the organizational context, reuse aims and practices, as well as the project structure and available resources.

PuLSE has been applied successfully in various different contexts for different purposes. Among other things it has proved helpful for introducing sound documentation and development techniques into existing development practices.

1.1.3 Goal

The goal of this report is to present the state of the art in product line technology and its relations to business process technology. As there is only little research work available that addresses the usage of product line technology in the area of business processes, the report develops some initial concepts for the fruitful combination of the two disciplines.

Copyright © Fraunhofer IESE 2006

1.2 Outline

The remainder of this report is structured as follows. Chapter 2 presents the principles of software product lines that are then used in Chapter 3 for the discussion of business process variants. Chapter 4 summarizes this report and presents the subsequent research steps of Fraunhofer IESE within PESOA.

Copyright © Fraunhofer IESE 2006

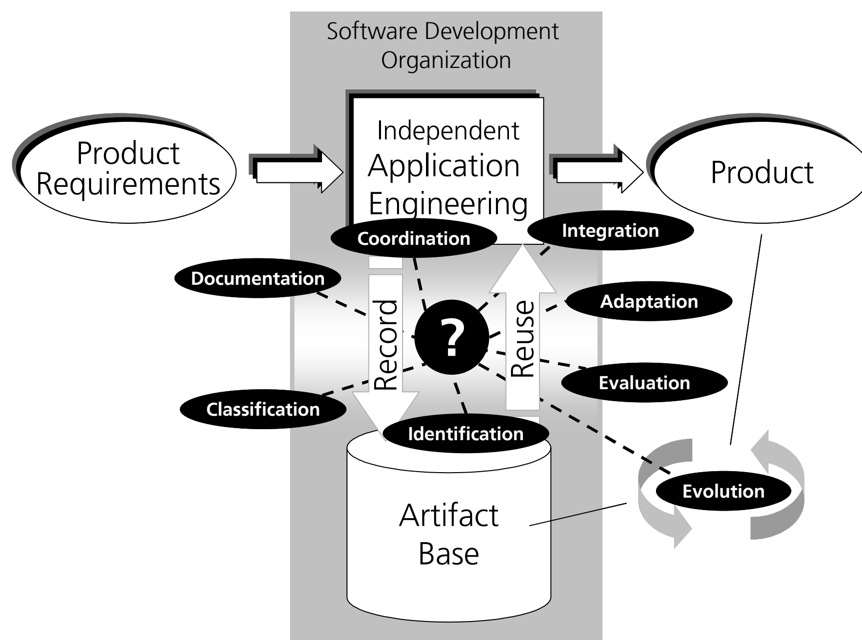
2 Principles of Software Product Lines

This chapter presents the principles that underlie software product line engineering. First, the principal software engineering approach that is followed in software product line engineering is given in section 2.1. Section 2.2 then presents the major concepts in product line engineering, namely commonalities and variabilities. The product line infrastructures that enable large-scale reuse are described in section 2.3.

2.1 Product Line Engineering

In general, product line engineering aims at the systematic development of a set of similar software systems by understanding and controlling their common and distinguishing characteristics. Product line engineering is an approach towards software reuse. A straightforward and commonly used approach to software reuse is independent application engineering where an organization develops each product as part of an independent application engineering project. The artifacts making up the resulting product are stored in an artifact base with the intent to reuse them in subsequent projects. Independent application engineering and its challenges are shown in Figure 3.

Figure 3. Independent Application Engineering.



Copyright © Fraunhofer IESE 2006

As many organizations have experienced, however, such a simple, straightforward approach usually does not achieve the expected improvements in reuse. The simple approach typically does not define a means for organizing and managing reusable artifacts in a way that effectively supports application engineering projects and thus does not typically make people promote reuse in an organization.

The challenges that an effective approach must cover to be successful (see [BR91]) are all related to the artifact base as depicted by the black ellipses in Figure 3. These challenges must be tackled by more advanced reuse approaches that aim at improving the role of reuse with respect to independent application engineering.

Domain engineering is such a reuse approach that pioneered the idea of planning and partially developing similar systems – systems in the same application domain – concurrently [Pri90]. Thereby an application domain is defined by the rough characterizations of the set of systems understood as being part of the application domain of interest.

Domain engineering analyzes an application domain, its abstract concepts, entities, and relationships in order to build a reference model for systems in the domain including domain-specific reusable artifacts. Thus, artifact subsumes all kinds of work products manipulated by development activities. Concrete applications are then constructed mainly by reusing the domain-specific artifacts, which represent the domain concepts or features required for the concrete application.

Domain engineering initially started with the Draco approach published by Jim Neighbours [Nei80, Nei89]. An overview of the genealogy of domain engineering methods can be found in [Lim98]. Overviews of, and comparisons between particular domain engineering methods can be found at numerous places in the literature, for example, in [Ara89, Mut97, Lim98, SS99, CE00].

Domain engineering tackles all the reuse challenges identified above with the concepts of the real-world application domain. Table 1 describes the solutions proposed by domain engineering, for each of the reuse challenges. Hence, the table characterizes domain engineering as a reuse approach according to the dimensions of the reuse taxonomy introduced by Krueger [Kru92]. The details of its four dimensions: abstraction, selection, specialization, and integration depend on the particular implementation of the general domain engineering approach in an organization.

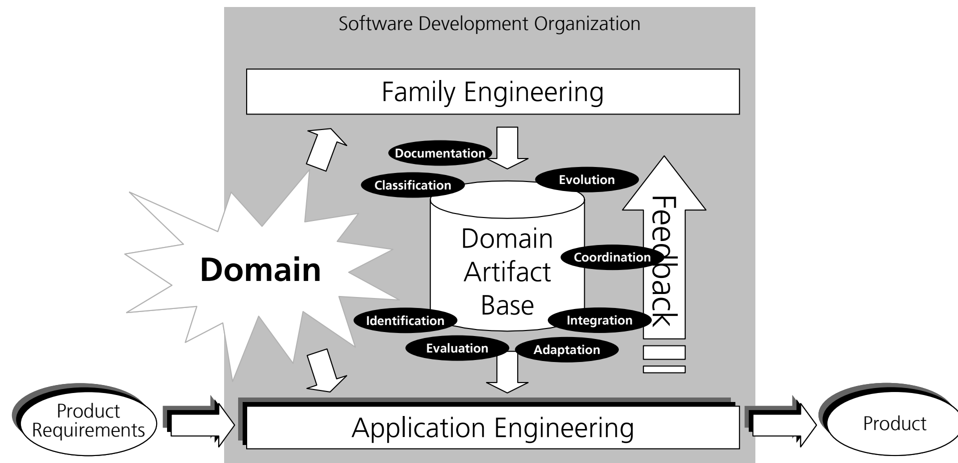
Copyright © Fraunhofer IESE 2006

Table 1. Software Reuse Challenges and Domain-Oriented Solutions.

Challenge	Problem	Domain-Oriented Solution
Documenta-tion	Each artifact placed in the artifact base must be documented to facili-tate its reuse.	The entities and relationships in the domain are used to document the reus-able artifacts.
Classifica-tion	All artifacts and associated docu-mentation must be structured ac-cording to a common classification scheme that optimally supports the reuse process.	Artifacts are classified according to the structure of the domain by domain ex-perts.
Identifica-tion	An identification mechanism (i.e., classification scheme) is needed to provide information available reus-able components.	Places that could benefit from reuse often correspond to problems in the domain of interest. Thus, searching the artifact base for solutions to the domain problem can identify reuse candidates.
Evaluation	When a set of potential reuse can-didates has been identified, the candidates must be evaluated with respect to adaptation and/or devel-opment effort.	Reuse candidates can be evaluated using the domain abstractions; differences are then expressed in terms of domain vari-abilities.
Adaptation	When a reuse candidate has been selected, it must be adapted or parameterized to fully match the actual requirements. In order to keep reuse efficient, effort spent on adaptation must be smaller than the development from the scratch.	The variability in the domain is explicitly modeled and documented for each reus-able artifact in the domain artifact base. Adaptation thus means for a significant part of the adaptation simply customiz-ing all points of variation in a clearly defined way with respect to the actual requirements.
Coordina-tion	An organization must coordinate concurrent application engineering projects to avoid identical adapta-tions within different projects.	The split of the development life cycle enforces feedback of the application engineering projects to the domain engi-neering activities.
Integration	Reused artifacts must be integrated with the application under devel-opment so as to remove conflicting assumptions about the environment and architectural mismatches.	A reference architecture simplifies inte-gration because the artifacts are built to be reusable for this architecture and any associated implicit assumptions concern-ing the domain.
Evolution	The maintenance of artifacts reused in numerous applications is more complex than for artifacts used only in a single application.	Domain engineering coordinates the processing of the many change requests originating from users of specific applica-tions in the domain. Since the mainte-nance of all reusable artifacts is primarily performed at the domain level, mainte-nance effort is integrated and minimized.

Copyright © Fraunhofer IESE 2006

Figure 4. Domain Engineering.



Domain engineering is shown in Figure 4. As can be seen there, domain engineering adds another main phase to the overall software life-cycle. In addition to application engineering, which is still responsible for building concrete products, a family engineering stage is performed in which the domain is analyzed and domain-specific artifacts are defined and constructed for reuse only. A product family is thus the subset of potential systems in a domain that contains the systems considered while the domain artifact base is constructed.

From an external point-of-view any software development organization delivers products based on product requirements. However, the key difference between an organization performing independent application engineering and a domain engineering organization can be clearly identified when domain engineering is viewed as an application-engineering approach that exploits the fact that organizations mostly perform more than a single application-engineering project over time. Domain-engineering organizations, therefore, explicitly analyze their application domains in addition to the construction of concrete products. Hence, domain engineering can be defined as a domain-analysis-based approach towards application engineering.

The artifacts that the product consists of are in an ideal case completely produced by reusing artifacts from the domain artifacts base that has been built up during the initial family-engineering activities. The reuse of artifacts, as well as the construction of artifacts for reuse, is thus an integral and explicit part of the overall approach. That is, software reuse is an inherent and central paradigm of software development organizations applying domain-analysis-based application engineering.

In general, there are three potentially complementary ways for reusing domain-specific knowledge. First, the knowledge is integrated into a domain-specific

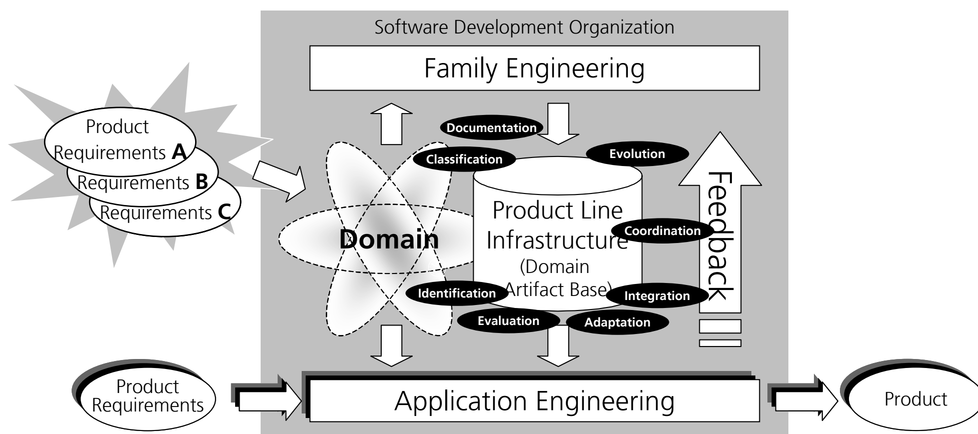
Copyright © Fraunhofer IESE 2006

language, or description technique, used to specify applications throughout application engineering. Second, the knowledge is captured in form of decisions that must be taken throughout the application-engineering process and thus are documented in the underlying process model. Third, the knowledge is captured in generic artifacts for which a generation procedure is defined to systematically transform these generic artifacts into concrete artifacts matching the context of a particular application. That is, the product model of the used products provides a means for capturing variability. The three alternatives span a domain of approaches for reusing domain-specific knowledge.

Although domain engineering represents a significant step forward in the support of reuse, practical experience has shown that the definition of the domain under consideration is problematic. When the domain is chosen to be too small, the domain model and the reference architecture fail to address important issues. As a result, significant changes will typically be required in future projects, undermining the value of the whole domain engineering effort. On the other hand, when the chosen domain is too big, the effort invested in domain engineering is higher than is really necessary, and the already significant investment involved in domain engineering goes beyond that which is cost effective. In short, although domain engineering provides a powerful set of solutions to the fundamental reuse problem identified previously, the success of a domain engineering effort is highly sensitive to the correct definition of the domain. If a domain is chosen to be too big or small, domain engineering can do more harm than good.

Product line engineering solves this problem by using only the characteristics of a finite number of concrete (existing, planned, or future) products to define the domain. Everything required by a concrete product is part of the domain - everything else is outside [DS98]. Figure 4 visualizes this concept of defining the domain through a set of products.

Figure 5. Product Line Engineering.



Copyright © Fraunhofer IESE 2006

The resulting domain is the subset of the application domain that corresponds to the concrete set of considered products. This subset is called the product line scope.

When the scope has been defined, there is no basic difference between product line engineering and domain engineering as described above except that the invested effort is applied in a more focused way. Because product line engineering defines the content of the effort by the requirements on a set of concrete systems, family engineering more efficiently produces an infrastructure beneficial to projects in an organization than is usually the case with traditional domain engineering, which completely analyzes an abstract application domain defined by fuzzy boundaries.

However, product line engineering does not completely avoid fuzziness and uncertainty. With respect to the problem of continuously changing requirements that many single-system projects face, requirements on future systems, which are key for a more concrete scope definition, can in reality typically also not be seen as fixed. The capabilities of competitor products, the technologies that can or must be supported, as well as the concrete and detailed needs of customers are unknown or at least uncertain for the future.

Product line scoping usually refers to the discipline of determining the bounds promising the best return on investment on a product line effort [Sch00]. The fuzziness of the resulting scope is related to the fuzziness introduced by the selection and the number of products taken into account, as well as on the level of detail considered during the scoping activities.

To summarize the historical evolution, the effort spent on finding a successful reuse approach has led to product line engineering consisting of a combination of intelligent scoping and efficient domain engineering. In other words, the success of product line engineering as a reuse approach depends on the quality of the defined scope. Unfortunately, a product line scope always contains some uncertainty; especially in the software context where technology changes fast and customer requirements evolve constantly. Consequently, any decision made during domain engineering could eventually turn out to have been wrong and the effort needed for its correction may be so large that the whole product line effort could be questioned. In these cases, there is no return of the investments and thus product line engineering is perceived as a non-successful reuse approach.

A way out of this dilemma is to accept the reuse infrastructure, as an imperfect, non-optimal but nevertheless effective tool for meeting business needs. If this view is accepted, an evolutionary approach towards product line engineering allows the infrastructure to be continuously adapted and improved and thus to meet the typically rapidly changing business needs determined by the products under development [TCO00].

Copyright © Fraunhofer IESE 2006

2.2 Product Line Concepts

This Section discusses the essential characteristics of product line engineering. From an abstract point of view, it is the concurrent consideration, planning, and comparison of similar systems that distinguishes product line engineering from single-system development. The intention is to systematically exploit common system characteristics and to share maintenance effort.

In order to do so, the common and the varying aspects of the systems must be considered throughout all life-cycle stages and integrated into a common infrastructure that is the main focus of maintenance activities.

Commonalities and variabilities are equally important: commonalities define the skeleton of systems in the product line, variabilities bound the space of required and anticipated variations of the common skeleton.

2.2.1 Commonality

Product line engineering is only useful when an organization develops several systems in one application domain. By definition, this implies that these systems have at least some characteristics in common, otherwise it would be difficult to view them as occupying the same domain. In a sense, therefore, the common characteristics of a family of products serve to characterize the domain. Typically, organizations limit themselves to the domain or domains that they have expertise in.

Commonalities are important for establishing a common understanding within an enterprise of the kinds of applications that it provides. The determination of whether a characteristic is a commonality or variability is often a strategic decision rather than an inherent property of the product family. For example, the execution platform can be a commonality when an organization decides to provide a solution for only one particular platform.

2.2.2 Variability

Variabilities are characteristics that may vary from application to application. One goal of the product line approach is to control the variabilities among systems in a family, that is, to minimize the number of unexpected adaptations and features within application engineering projects by planning, in advance, for future requirements. Therefore, markets and customer behavior must be observed and analyzed to get good predictions of future domain requirements and trends.

Copyright © Fraunhofer IESE 2006

Variabilities that an organization wants to support are explicitly modeled, documented, and integrated with the product line infrastructure. In general, all variabilities can be described in terms of alternatives. At a coarse-grain level, one artifact can be seen as an alternative to another artifact. Then during application engineering, the artifact that best matches the context of the system under development is selected. Although simple in theory, providing an effective representation of the variabilities in a product family is an important factor in successfully managing a product line infrastructure.

Simply identifying and modeling alternatives among the products in a product line does not define what characteristics are associated with what products, as well as what dependencies and interrelationships exist among variabilities. This information must also be captured, which is often the role of a decision model. Essentially a decision model consists of decisions that relate user visible options to specific system characteristics. Its goal is to support the evolution of the product line infrastructure and to guide application engineers in using the infrastructure while building new applications.

2.3 Product Line Infrastructures

One of the core elements of a software product line is an artifact base (or in particular a product line infrastructure). The name artifact base stems from the basic reuse model where all kinds of artifacts produced in a project are stored and, if possible, retrieved and reused in subsequent projects. Storage and retrieval can be handled by ordinary file systems or simple databases but if the artifact base is optimized for the support of application engineering in the context of product line engineering, an artifact base is customized to the needs of an individual product line and, thus, it is called a product line infrastructure.

As depicted in Figure 5, a product line infrastructure is an internal part of a software development organization. The product line infrastructure represent information on the organization's planned and delivered products in an integrated form. From the external customer's point of view, only information on individual products is visible and the organization - as any other software development organization - develops a series of products each fulfilling certain customer requirements.

2.3.1 Product Line Information

The core idea of product line engineering is to analyze a set of systems and exploit their commonalities systematically rather than developing system by system individually. This implies that information in a product line context is mainly concerned with multiple systems. One possible way to structure such information is to compare a set of systems but to keep the information on each indi-

Copyright © Fraunhofer IESE 2006

vidual system separately visible. Such a separation of system information is typically required at two places in the product line life-cycle:

- First, during family engineering when the scope of the product line infrastructure is analyzed to plan a product line infrastructure. There, each system to be considered as part of the product line is characterized and compared to all other systems of the family.

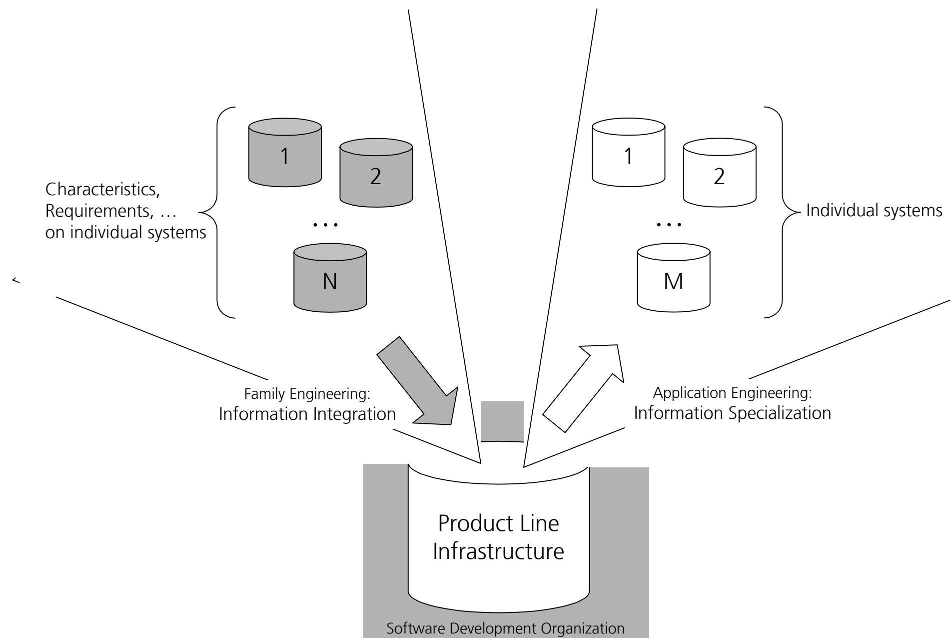
- Second, during application engineering where for particular customers, only his/her system is relevant. Even though these systems are (partially) derived from the product line infrastructure, information discussed with, validated by, and delivered to the customer only contains information on a particular system.

Product line engineering, therefore, manipulates information on systems in two ways as depicted in Figure 6. Family engineering analyzes information on single systems, integrates it by consideration of commonalities and variabilities, and stores the integrated information as part of the product line infrastructure. Application engineering uses the integrated information and specializes it according to the needs of a particular customer.

In the latter case, information about single systems is relevant and thus there is nothing product-line-specific. In the former case, again only information on single-systems is captured but in a second step (e.g. product line scoping) multiple systems are compared. The performed comparison may be product-line-specific but the artifacts that capture the information are similar to artifacts also used for market surveys or product portfolio descriptions in a non-product-line context.

Copyright © Fraunhofer IESE 2006

Figure 6. Information in the Product Life Cycle.



Examples of this kind of artifact are the product map, which relates features and systems in a tabular form, as used by [Sch01] throughout scoping, or the requirement documents used by the viewpoint-oriented domain requirements definition method (VODRD) that put the stakeholders' viewpoints on multiple product line members side by side [MKH98].

In the remaining and larger part of the product line life cycle, the characteristics of systems are handled and captured in an integrated way. That is, information captured by artifacts focuses on an application domain in general, its commonalities and variabilities, rather than on comparing common and varying characteristics of particular systems. Of course, mappings between the integrated product line information on the one side and each system that has either been input to product line planning or been derived from the product line infrastructure, on the other side, is needed. This is particularly important because this traceability information enables the sharing of maintenance and evolution effort among product line members in the long run. Such a mapping is defined with a decision model that is also part of the product line infrastructure. When product line information is integrated, commonalities and variabilities are part of one artifact, a product line artifact.

A product line artifact is an artifact that captures product line concepts such as commonalities or variabilities in an integrated and explicit form. A product line

Copyright © Fraunhofer IESE 2006

artifact that captures no variabilities is identical to an artifact used in a single-system context.
The difference between a product line artifact and a non-product line artifact is that a product line artifact not only contains run-time variability but also development-time variability that expresses the difference between products in a product line. Because run-time variabilities are an inherent part of software, they are already handled effectively by traditional techniques. Therefore, to capture run-time variability, in general, no special artifacts are required.
Product line engineering, in contrast, is more concerned with variabilities among systems that are typically resolved before a software system is loaded onto its final execution environment. The problem is that there is no strict boundary between development-time variabilities and run-time variabilities as described above. Deciding whether a variability should be realized as a choice during development time, or whether it should be built into a system for resolution at run-time, is a strategic decision that can have a large bearing on the success of a product.

Independent of the technical realization of variability, the key difference between single systems' run-time variability and variability in an application domain (i.e., development time variability) is that the former is an inherent part of the final software system, while information on the latter must be explicitly controlled to effectively and successfully manage a product line and use it to create systems. Basically all activities in the product line life-cycle require information on what varies from one system to another, what motivates these variations, and where these variations impact software solutions and related artifacts.

This variability information is important all over the product line life cycle. Even if there is an agreement on the importance of variability information and the need to handle it differently from usual run-time variability, the right level of detail and presentation style of (development-time) variabilities must still be determined. Because they have always had to deal with run-time variabilities, most graphical modeling languages already have rudimentary facilities for representing them. Often, variabilities are handled by a combination of generality and constraints. Information is typically presented at a level of generality that covers all possible run-time variations, and is accompanied by textual constraints that specify which of the many possible combinations are acceptable.

In theory, development-time variabilities could be handled in the same fashion as run-time variabilities (as mentioned above, development-time variabilities can be realized as run-time variabilities instead) this basic approach could still be used to handle product lines. In other words, information could be generalized to the point where it makes statements that accommodate all members of the family, and could be supported by constraints describing how the general information changes among family members. However, this approach has two major drawbacks. First it would mean that the information about variabilities is

Copyright © Fraunhofer IESE 2006

actually captured in separate constraints rather than integrated with the artifacts it relates to. This defeats the whole object of easily accessing and understanding variability information because information must always be intellectually related to the attached constrains. Second, and more importantly, it means that variabilities that distinguish family members in a product line are mixed up with the run-time variabilities that are "common" to all members of the product line. In short, it suppresses the very information that is essential for effectively understanding and using a product line.

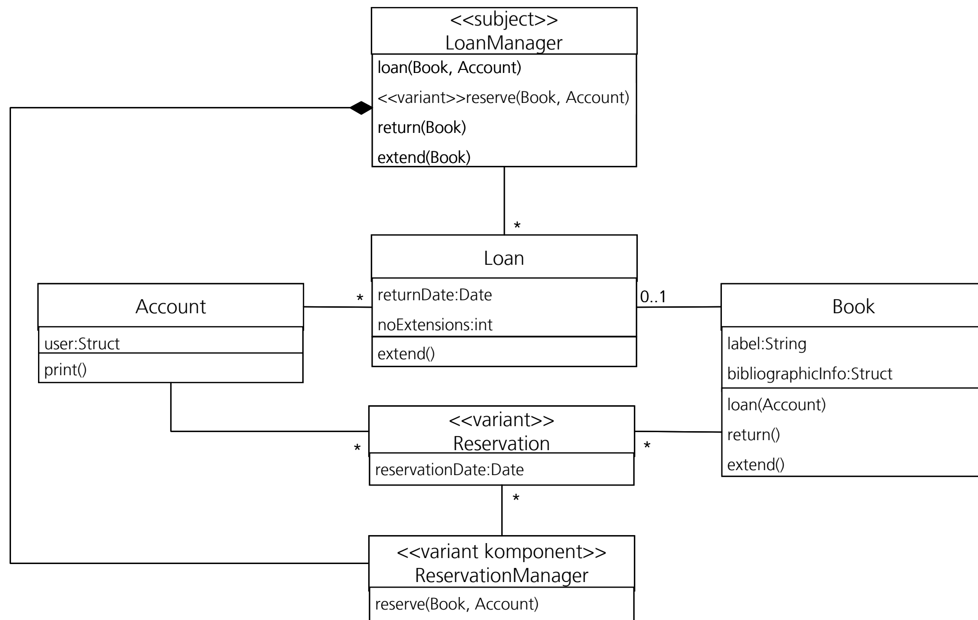
As motivated above, information on variability is key to activities all over the product line life-cycle. Hence, the variability information must be easy to access and understand. Therefore, variability information is captured and documented explicitly and it is integrated with information on commonality and other variability as well. How this is done in the PuLSE method is presented in the next section.

2.3.2 Elements of Product Line Infrastructures

A product line artifact is any kind of artifact that captures information about the systems of a product line. The artifacts are built to be reused when systems in the product line are developed. As described above, there are two types of information captured: information that is valid for all systems in the family (i.e., commonalities) and information that varies from system to system (i.e., variabilities). Variation points in the artifacts denote variability by showing the different possible alternatives.
The variation points in a product line infrastructure must be resolved when a particular product line member is specified. Resolving variation points means that parts of the artifact are specialized, included, or excluded.
Figure 7 shows a product line artifact, a UML class diagram. There, variation points are represented using the stereotype concept of the UML. In this way, generic assets can be modeled in a UML compatible way (the UML itself does not provide any means to capture variability). Figure 7 describes the structure of a loan manager component that is used in a library system to manage loans of library users. Some libraries supported with a product line member provide the possibility for a user to reserve an item in the case that it is already loaned to someone else. Other libraries do not provide reservation facilities. Therefore, reservation is a variability that is related to three variation points in the class diagram (the two classes Reservation and ReservationManager, and the method reserve of the LoanManager).

Copyright © Fraunhofer IESE 2006

Figure 7. Class Diagram for a Generic Loan Manager Component.



There is typically a large number of variation points in the assets of a complete product line infrastructure. Consequently, it is, even for experts, hard to control the rationales for each variation point, as well as the complex interrelationships and dependencies among them.

To support the intellectual control, a decision model, which captures this kind of domain knowledge, is built on top of the variation points.
A decision model consists of a decision hierarchy, which is grounded on simple decisions that capture the rationale and the possible choices for a single point of variation. Dependencies among decisions are explicitly captured by constraints. Usually, additional decisions are introduced. These decisions are not directly related to a point of variation of an asset but they represent domain variability at a higher level of abstraction. This higher abstraction level is related to sets of interdependent variation points.

The following table shows the decision model for Figure 7.

Copyright © Fraunhofer IESE 2006

Table 2. Decision Model for Resolving Variation Points.

ID	Question	Variation Point	Resolution	Effect
1.1.	Is reservation facility needed?	operation LoanManager.reserve()	Yes (default)	remove stereotype <<variant>>
			No	remove operation LoanManager.reserve()
		class Reservation	Yes (default)	remove stereotype <<variant>>
			No	remove stereotype <<variant>>
		class ReservationManager	Yes (default)	remove stereotype <<variant>>
			No	remove stereotype <<variant>>

A decision consists of a unique id for identification, a question that is asked for resolving the decision, the variation point it is related to, a set of possible resolutions (of which one is identified as the default resolution), and a description of the effects the resolutions have on the diagrams. In KobrA, OCL (Object Constraint Language - a notation to describe UML diagrams) [WK99] can be used to describe the effects of a decision in addition to textual descriptions as they are used above.

A product line instance (or product line member) is a system that is developed with the reuse of product line artifacts for a particular customer. It is the output of application engineering, the process of developing specific product line members. The tailoring of a product line asset while reusing it is a two-step process: first, its instantiation in the space of supported variabilities and, second, its extension with aspects that are not supported by the product line asset but that are required by a particular customer.

The decision model drives the application engineering process, that is, while traversing the decision hierarchy, decisions are resolved — one at a time. When a decision is resolved that constrains variation points of a product line asset, the artifact is instantiated accordingly, that is, the variation point is removed and replaced by the concrete realization that corresponds to the selected resolution. The resolution process stops when all simple decisions are resolved. The resulting asset instance is the variant of the variants supported by the product line infrastructure that is closest to what is required in the specific context. Often, even the closest variant is not exactly what is required and, thus, further modifications are necessary.

Copyright © Fraunhofer IESE 2006

The specific variants that are obtained by resolving the variation points of the generic LoanManager component are depicted in Figure 8 and Figure 9.

Figure 8. Specific LoanManager Component without Reservation Facility.

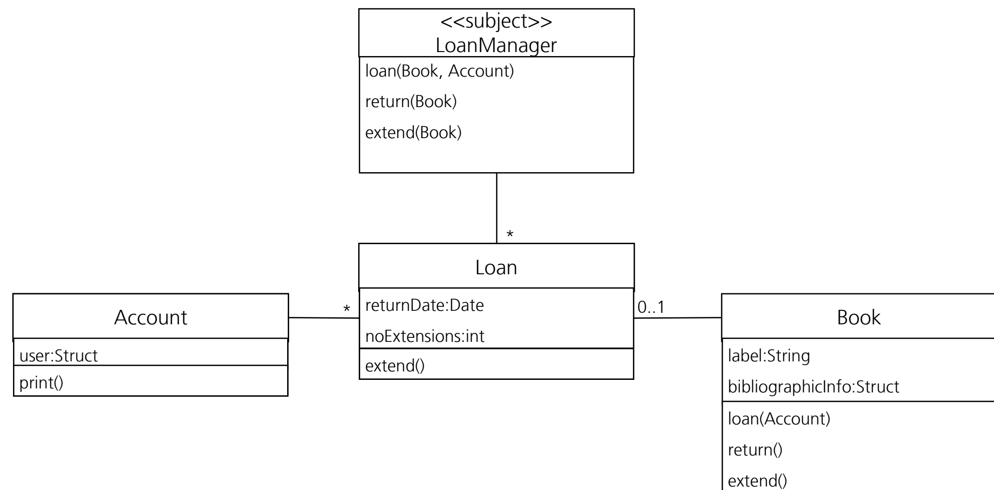
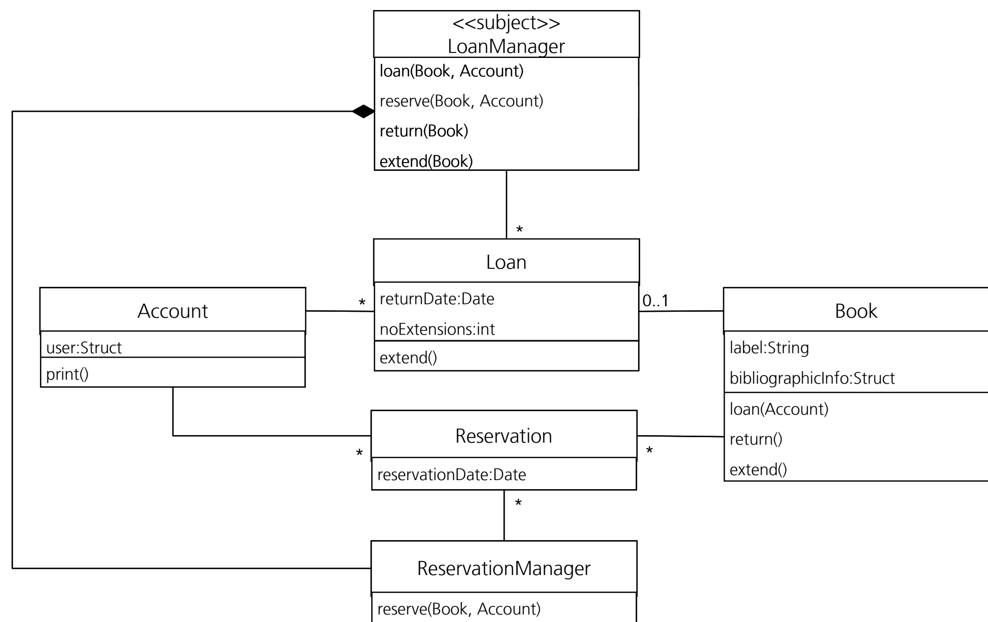


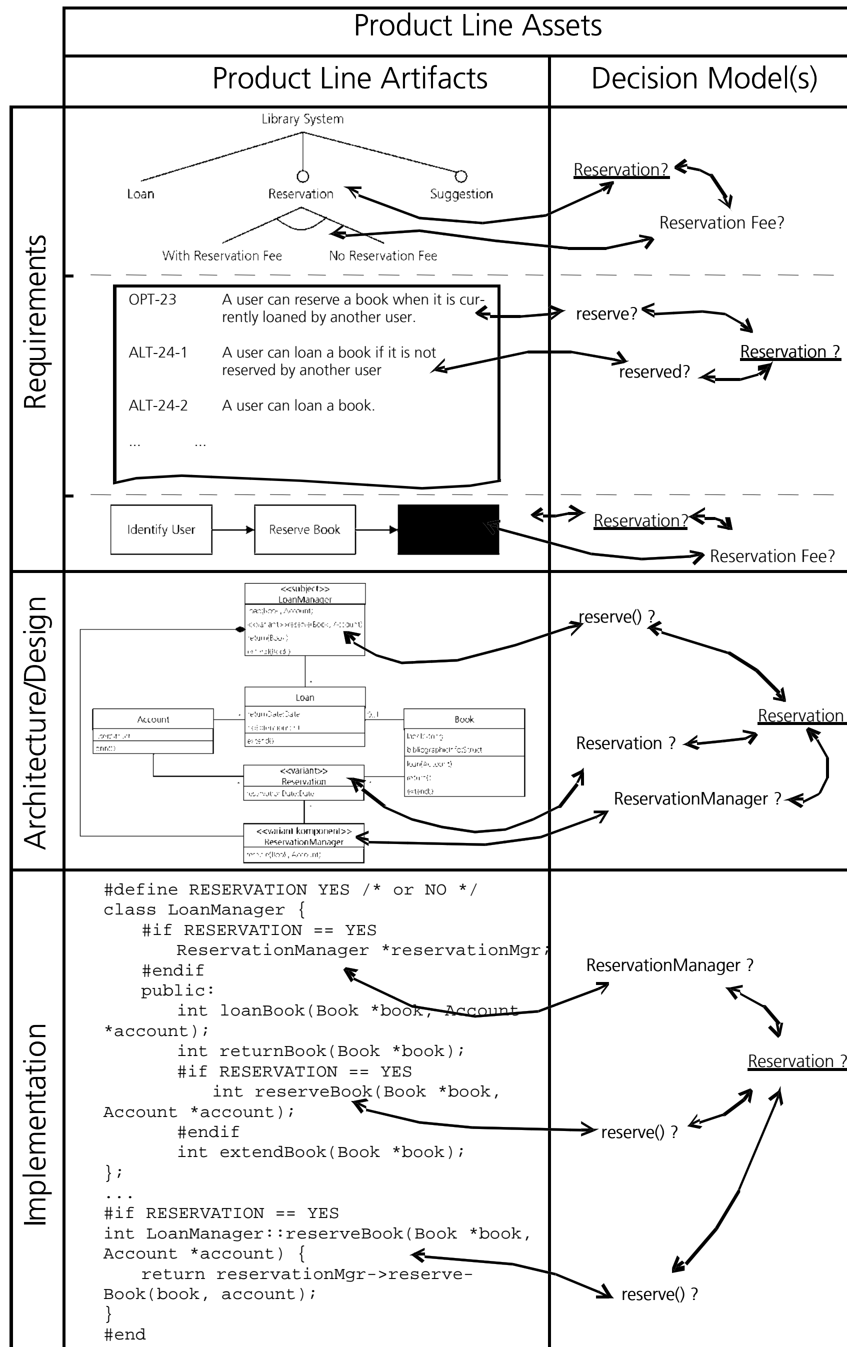
Figure 9. Specific LoanManager Component with Reservation Facility.



Potentially all artifacts used to document software can be extended to be product line artifacts. Figure 10 depicts one instance of a product line infrastructure in a table-like structure.

Copyright © Fraunhofer IESE 2006

Figure 10. Overview of an instance of a product line infrastructure.



Its rows correspond to life-cycle stages, such as requirements, architecture, design, or implementation. Its columns split each row into two pieces: on the left-hand side, product line artifacts, and on the right-hand side, the related deci-

Copyright © Fraunhofer IESE 2006

sion model, that is the information that allows the artifacts' variabilities to be controlled. The shown instance of a product line infrastructure is an infrastructure for the library system example used above. The modeled variability is again the optional reservation feature. The decision model allows this optional feature to be traced throughout all life-cycle stages and all variation points related to reservation to be identified within the product line artifacts that are part of the product line infrastructure.

The product line artifacts shown in Figure 10 are (from top to bottom): feature model, textual requirements document, a business process model, a UML class diagram, and a C-source-code file. Each of the shown product line artifacts captures product line information and also some variability, such as optional features, alternative requirements, optional activities of a business process, alternative classes, or optional source-code elements.

The variant parts of a product line artifact yield variation points within an artifact that must be resolved later to specialize the artifact to the needs of a particular product context

2.3.3 Processes as Variability Driver

In principle all non-generic artifacts used in software engineering can be used in a product line infrastructure. To this end, they need to be made generic artifacts. Details on how to achieve that can be found in [Mut02].

Within enterprise applications, business processes are a major driver for variability. For example, a process variant may be required for a different customer or just as a natural evolution of the process over time (details see section 3.3).

The next chapter gives an overview of the state of the art regarding the engineering of business process variants.

Copyright © Fraunhofer IESE 2006

3 Process Variants

3.1 Origins and System Evolution

Historically, the term business process emanated from the area of Enterprise Resource Planning - ERP. Here, the business process was introduced in order to describe information flows in the enterprise [Sch97]. Concerning ERP systems, business processes have now been implemented for almost 30 years now. From a system evolution point of view, however, different currents have to be distinguished. The most important ones are discussed in the following:

Enterprise Resource Planning

The first systems for enterprise resource planning emerged in the 70s, for example SAP and ORACLE [Sch97]. The initial motivation of these systems was to cover recurring functionality in business administration such as production planning, finance, accounting and human resources. In most cases, this lead to architectures with functional building blocks as main characteristic (Figure 12). In these architectures process logic is rarely managed as an isolated artifact but is rather spread or "distributed" over different modules.

Workflow Management Systems

The first workflow management systems emerged in the late 80s during the wave of "office automation" [JBS97]. The motivation was to improve typical office workflows via automation. In most cases the workflow is defined in a descriptive notation that is then processed and monitored by a so-called workflow execution engine. In this way workflow systems focus on the workflow that consequently forms an explicit part in the system architecture. This is a contrast to the traditionally function and module - oriented ERP systems where processes exist on a more implicit level (see above). Yet another difference between workflow systems and ERP systems is that workflow systems traditionally focus on documents whereas ERP processes operate on relational data in databases. Over the time, however, ERP systems have partially adapted a more explicit workflow orientation.

Business Process Management Systems – BPMS

A new emerging trend since the late 90s is the application of the workflow paradigm for the integration of different enterprise information systems [DGH03]. This renaissance of the workflow paradigm can be seen as a combination of classical ERP systems with the workflow paradigm. Here business

Copyright © Fraunhofer IESE 2006

processes are explicitly defined in a central repository and participating systems are linked into the process at defined stages. This new type of systems is often referred to as Business Process Management Systems. Because of its system integration focus it is often discussed in the context of Enterprise Application Integration.

Figure 11 summarizes the different motivations and architectural properties of the discussed systems. From an evolution point of view, there is a clear trend for increasing process-orientation, integration and handling of the variation in business processes.

Figure 11. Evolution of Business Process Oriented Systems.

	1980	1990	2000	PESOA
Caracteristics	**ERP Systems**	**Workflow Systems**	**BPMS**	**BP Variability**
Motivation / Architectural Driver	**Functionality for Business Administration**	**Workflow for Workdocuments**	**Process oriented Integration of Business Appl.**	**Business Process Variability**
Dominant Design Methodology	**Functional Decomposition**	**Isolation of Process Logic (Separation of Concerns)**	**Isolation of Process Logic, Integration**	**Variant Management, Product Line Methodology**
Architectural View(s)	**Business Components**	**- Process**	**- Process - Integration**	**- Process - Variation - Integration**
Main Characteristics	**Functional Building blocks (e.g. SAP: FI, HR, SD, PM)**	**Explicit Workflow and Workdocuments, Single System (no integration)**	**Process-centric Integration of Business Applications**	**Product Line Architecture**
Architectural Manifestation of Process	**Implicit: Process Logic is spread over Source Code**	**Explicit: Process Repository and Process Execution Engine**	**Explicit: Process Repository and Process Execution Engine**	**Explicit: Process Repository and Process Execution Engine**

Business Process Aspect and Process-Oriented Views

Even though the systems have specializations, they all share the aspect of a business process or workflow. A process-oriented view, however, may be used for the visualization of any flow-oriented aspect.

For example, a "technical process" like the initialization sequence of business objects during the start up of an application may also be represented in a process-oriented view. This means that a process-oriented view can be used to visualize different aspects – such as business processes or specific technical processes (e.g., an object initialization sequence).

Copyright © Fraunhofer IESE 2006

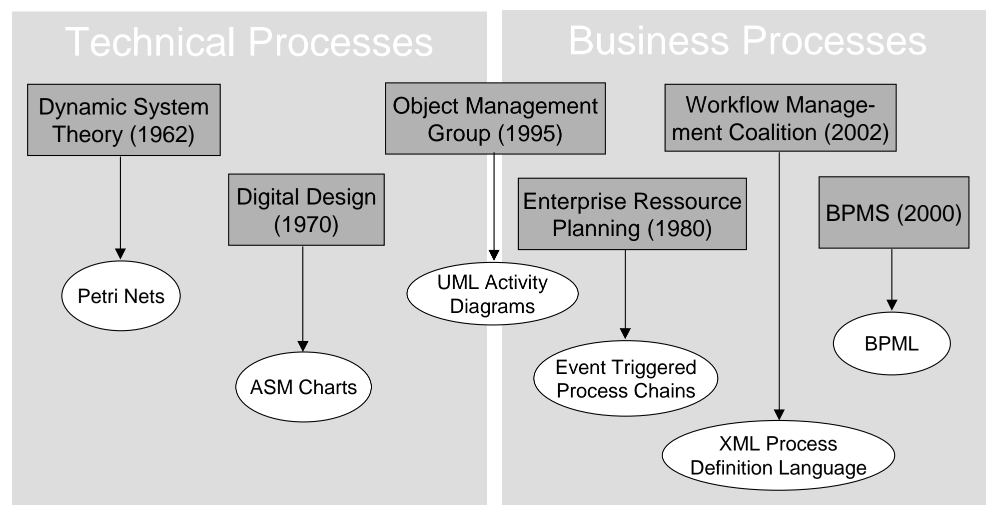
An aspect may require the focus on specific issues – for example, the representation of user interaction in a business process. This is the reason why specialized views and modeling techniques have been developed over time (see section 3.2).

PESOA deals with business processes as well as with technical processes. Therefore the term “process” will be used in the following for denoting any kind of process – business or technical.

3.2 Process Modeling – Standards

In parallel to the evolution of systems with process-orientation, an abundance of standards and technologies for process modeling and process implementation has evolved [Bal00, PW87, Sch97, JBS97, WFM02]. Figure 12 provides an overview of the most popular standards including a rough segmentation into main application areas (technical or business processes).

Figure 12 Origins and Evolution of Standards for Business Process Description.



The interesting issue when comparing the different standards is that the used core elements are often similar, if not identical. This strong commonality motivates for the isolation of a common meta-model that could be used as unified basis for the description of process variants (see section 4).
A complete comparison of the different standards goes beyond the scope of this report. Exemplary we focus on a brief comparison of three standards:

- **UML Activity Diagrams** represent that part of the Unified Modeling Language that deals with modeling business processes [Oes+03].
- **Event Triggered Process Chains** represent the business process modeling technique that is used within ARIS [Sch97].

Copyright © Fraunhofer IESE 2006

- **The Workflow Process Definition Standard** defined by the workflow management Coalition [WFM02].

Each standard is based on a specific meta-model with core elements and their relation to each other, e.g. [WFM02, p. 8]. Table 1 provides a comparison between the core elements of the three standards - activities, transitions, states and decisions [Sch97, [Oes97], [WFM02, pp. 8-11]. The commonalities between the different approaches are striking.

Table 3. Corresponding Core Elements in different Process Meta Models.

Activity Diagrams (UML)	Event-triggered Process Chains	Workflow Process Definition
Activity	Function	Workflow Process Activity, Atomic Activity.
Transition between Activities	Transition between Functions and Events	Transition between Activities
Transition between Activity and Object State	Event	Transition information or Flow Control Conditions
Decision	Decision	Condition

3.3 Need for Process Variants

The main reasons for business process variants in practice are industry-specific process requirements and continuous changes in business organizations and collaborations [Sch97].

Industry-Specific Process Requirements

It is a mere fact that different industry sectors require specific business processes. E.g., the order fulfillment process of an industrial material supplier contains a process step *Availability Check* in order to provide the customer with information about the concrete availability and delivery time for the ordered goods. In contrast, a simple online order and delivery restaurant that offers its meals within a guaranteed delivery time will not require a process step *Availability Check*. The crucial point is that the two business processes are not totally different. Rather they represent variants of the same generic business process *Order Fulfillment* (see section 3.4).

Changes in Business Organizations and Collaborations

The business processes of companies are subject to continuous changes. Typical drivers in this area are organizational changes (e.g., a new sales and distribution

Copyright © Fraunhofer IESE 2006

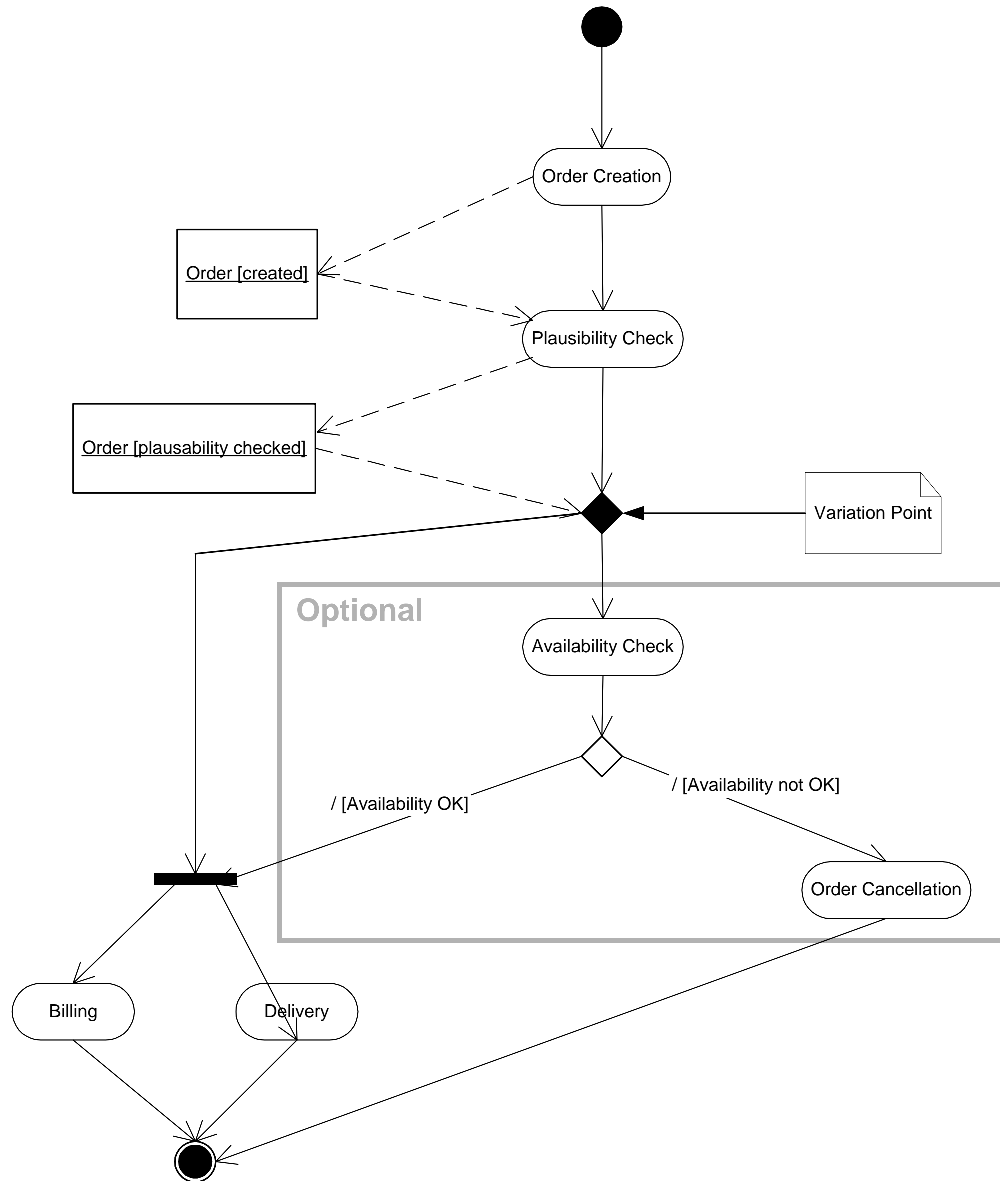
structure), new types of collaboration with business partners (e.g., collaboration with suppliers in a supply chain) and new legal regulations (e.g., the deregulation of the German energy market) [Sch+03, DGH03].

3.4 Modeling Process Variants

Up to now only little work has addressed the modeling of variation in business processes. The few existing examples express variation via the common concept of variation points. Figure 13 illustrates the usage of a variation point (which is depicted as black rhomb) for the modeling of a generic order fulfillment process using UML.

Copyright © Fraunhofer IESE 2006

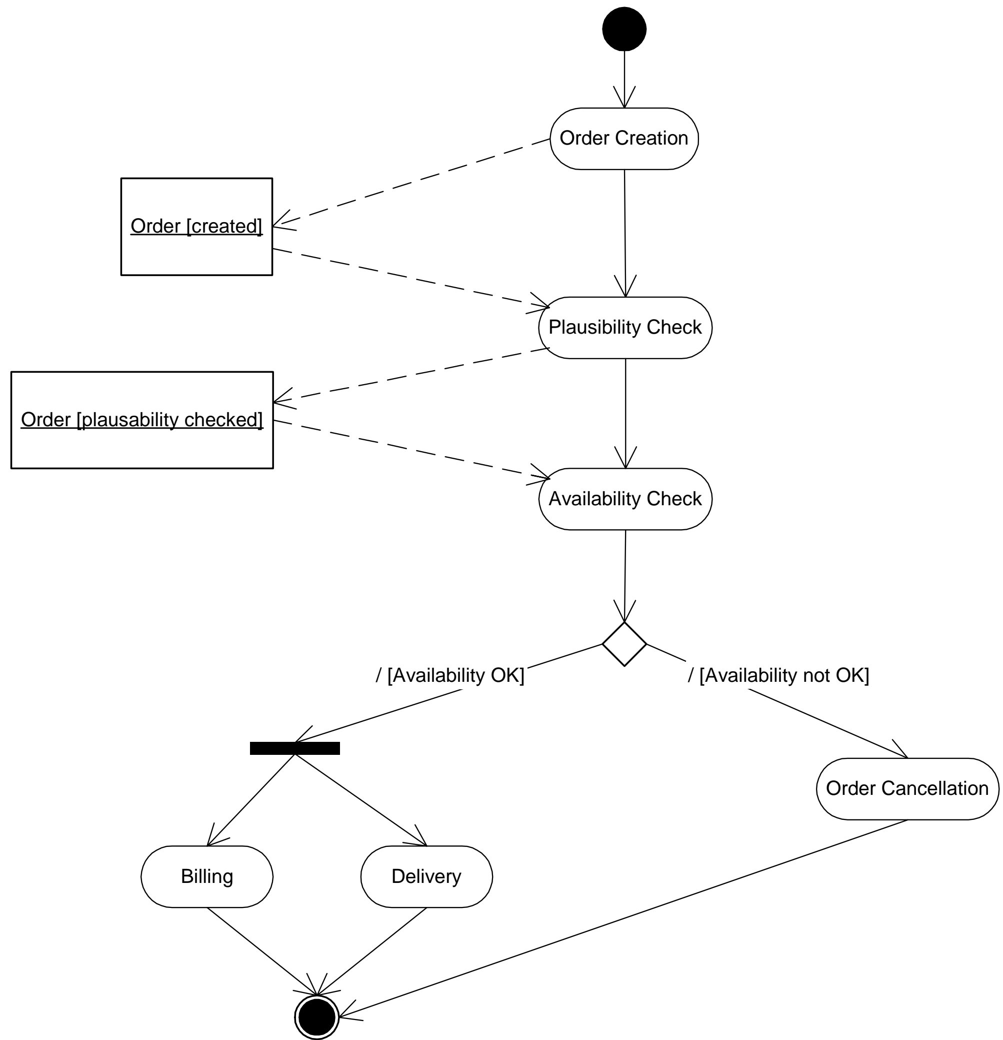
Figure 13. Generic Business Process "Order Fulfillment" with Variation Point.



In the example, the activity *Availability Check* (as well as the framed succeeding steps) is optional. Consequently there are two variants that can be instantiated from the generic process - a variant with Availability Check (Figure 14) and one without Availability Check (Figure 15).

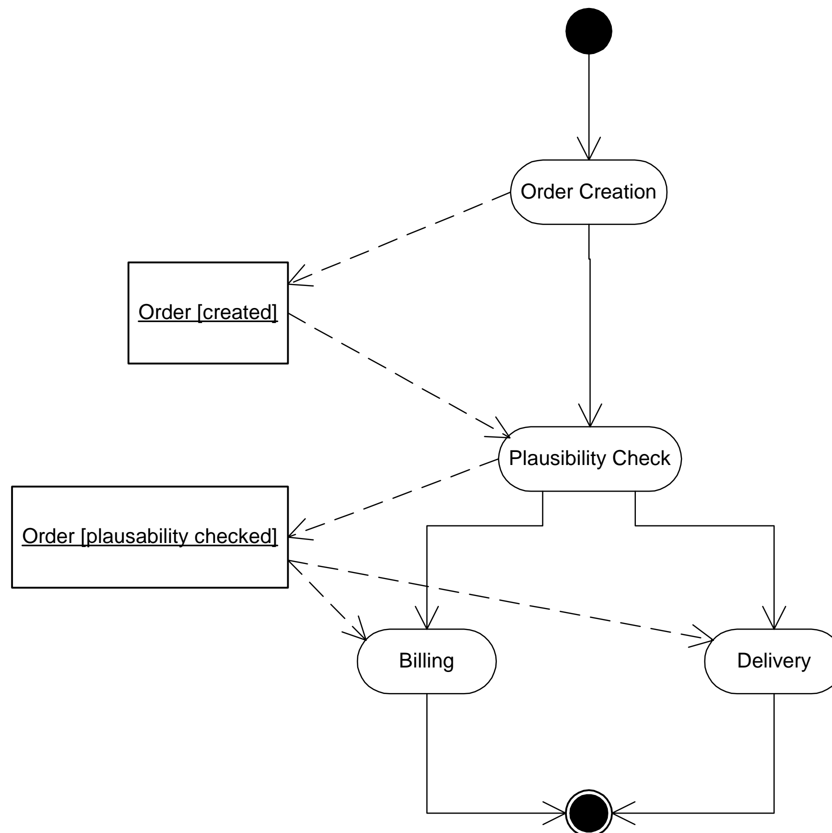
Copyright © Fraunhofer IESE 2006

Figure 14. Specific Order Fulfillment Process with Availability Check.



Copyright © Fraunhofer IESE 2006

Figure 15. Specific Order Fulfillment Process without Availability Check.

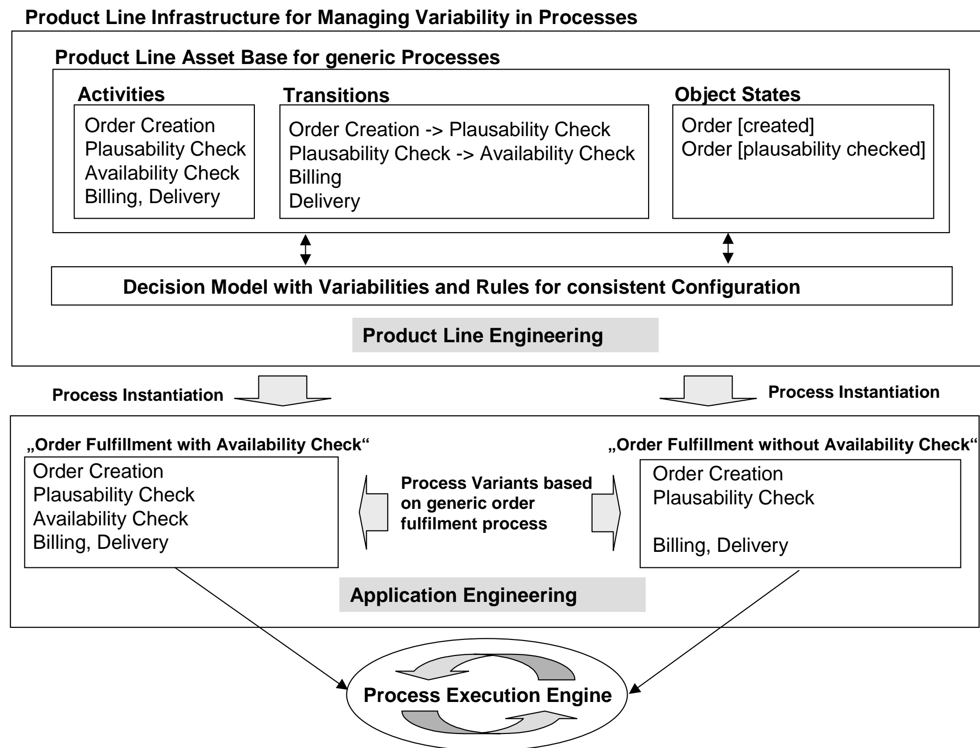


3.5 An Infrastructure for Managing Process Variants

Figure 16 sketches an exemplary infrastructure that supports the management of variability in business processes. The presented architecture follows the product line principles presented in chapter 2. The product line part contains all available assets for building a business process. Using the fundamental elements for process modeling in UML, the assets are typed into activities, transitions and object states. In addition, the product line infrastructure contains a decision model that defines possible variabilities and related consistency rules. For example, the information that the activity *Availability Check* is optional is defined here.

Copyright © Fraunhofer IESE 2006

Figure 16. Product Line Infrastructure for Managing Variability in Business Processes.



The presented approach has the following advantages:

- **Flexible Management of Variability in Business Processes.** This is the quality with the top interest for PESOA. As suggested in Figure 16, the management of business process variants is achieved with a typical product line software architecture that comprises generic process elements and that uses a decision model for instantiating specific process variants.
- **Isolation of Process Logic improves Changeability.** The clear separation of process logic from application logic simplifies and facilitates change management on process level. Business process logic in traditional ERP systems is often mixed with application logic what makes it often difficult and tedious to modify processes.
- **Flexibility for Integration.** The interfaces to external systems that are used within the process can easily be identified and exchanged with alternative components. This facilitates the integration and/or replacement of new components to/from the process.

Copyright © Fraunhofer IESE 2006

4 Conclusion and Outlook

This report presented the essentials of software product line technology, the leading approach for software mass customization, and analyzed its application for managing variability in the area of business processes.
The analysis shows that process-oriented software domains can benefit from product line technology, in particular concerning the efficiency and consistency with which variations in business processes can be managed.
Concretely, the transfer of product line technology to process-oriented software domains yields a process-oriented product line infrastructure that supports and facilitates the management of process variability.
Using such an infrastructure, software engineers create concrete process-variants by selecting and combining the process entities from an asset base with generic process elements. A decision model helps to resolve the existing variation points in a consistent way.

The presented state of the art in software product line technology and management of business process variability lead to an initial sketch for the design of a process oriented product line architecture.

The development of a consistent platform for process family engineering, however, requires further research regarding the following product line issues:

- **Domain Engineering and Business Process Modeling.** The existing domain engineering approaches will be analyzed concerning their ability for the modeling of generic business processes.
- **Product Line Engineering and Project Management.** Compared to single system development, the creation of a process-oriented product line requires different project management capabilities. The state of the art in project management for product lines and their specific adaptation towards process-oriented product lines will be analyzed here.
- **Asset Scoping.** The identification of reusable components is a crucial step in all product line approaches. PuLSE has developed scoping instruments that support the proper identification and selection of reusable components. However, these instruments will need to be adapted towards the specific requirements in a process family.
- **Process Configuration and Process Meta-Model.** The proper instantiation of specific processes from the process family should be based on a common business process meta-model. The obviously large intersection between the different modeling standards (section 3.2) motivates to distill a unified process meta-model.

Copyright © Fraunhofer IESE 2006

- **Decision Models and Configuration Technologies.** The process instantiation can be characterized as a configuration activity where complex decisions have to be resolved in a consistent way. Here, the usage of configuration technologies is promising. Existing configuration technologies will be surveyed, selected and adapted to the process-instantiation needs of within a process family platform.
- **Integration of Domain Engineering with Process Modeling.** This part will develop the methodology for applying domain engineering with process modeling.
- **Project Management and Process Family Engineering.** Based on the analysis of project management guidelines within product lines (see above), concrete guidelines will be derived for the efficient implementation of the process-oriented product line methodology in practice.

Copyright © Fraunhofer IESE 2006

List of Abbreviations

ARIS	Architektur Integrierter Informationssysteme
BPML	Business Process Markup Language
BPMS	Business Process Management System
EAI	Enterprise Application Integration
ERP	Enterprise Resource Planning
ETPC	Event Triggered Process Chains
IESE	Institute for Experimental Software Engineering
OCL	Object Constraint Language
PESOA	Process Family Engineering in Service-Oriented Applications
PuLSE	Product Line Software Engineering
UML	Unified Modeling Language
VODRD	Viewpoint-Oriented Domain Requirements Definition Method
WFMC	Workflow Management Coalition

Copyright © Fraunhofer IESE 2006

References

[Ara89] G. Arango. Domain Analysis - From Art Form to Engineering Discipline, in Proceedings of the Fifth International Workshop on Software Specification and Design, Sept. 1989.

[Bal00] H. Balzert. Lehrbuch der Software-Technik. Bd 1. Software-Entwicklung. 2. Aufl., Spektrum. 2000.

[BFK+99] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. PuLSE: A methodology to develop software product lines. In Proceedings of the Symposium on Software Reuse (SSR'99), Los Angeles, CA, USA, May 1999.

[BR91] V. Basili and D. Rombach, Support for Comprehensive Reuse, in Software Engineering Journal, Sep. 1991.

[CE00] K. Czarnecki and U. Eisenecker. Generative Programming - Methods, Tools, and Applications, Addison-Wesely, 2000.

[Cha02] G. J. Chastek (ed). Software Product Lines. Proceedings of the Second International Conference (SPLC2), San Diego, California, USA, August 2002.

[CN02] P. Clements and L. Northrop. Software Product Lines. Practices and Patterns. Addison-Wesley, 2002.

[DGH03] S. Dustdar, H. Gall, M. Hauswirth. Software-Architekturen für Verteilte Systeme. Springer 2003.

[Don00] P. Donohoe (ed.) .Software Product Lines - Experience and Research Directions. Proceedings of the First International Software Product Lines Conference (SPLC1), Denver, Colorado, USA, August 2000.

[DS98] J.-M. Debaud and K. Schmid. A Systematic Approach to Derive the Scope of Software Product Lines, in the Proceedings of the 21st International Conference on Software Engineering (ICSE), IEEE computer Society, 1998.

[JBS97] S. Jablonski, M. Böhm, W. Schulze. Workflow Management – Entwicklung von Anwendungen und Systemen. Dpunkt Verlag 1997.

Copyright © Fraunhofer IESE 2006

[Kru92] C. Krueger. Software Reuse, ACM Computing Surveys, vol. 24, no.2, June 1992.

[Lim98] W. C. Lim. Managing Software Reuse - A Comprehensive Guide to Strategically Reengineering the Organization for Reusable Components, Upper Saddle River: Prentice Hall PTR, 1998.

[MKH98] M. Mannion, B. Keepence, and D. Harper. Using Viewpoints to Define Domain Requirements, IEEE Software, Jan. 1998.

[Mut97] D. Muthig. Supporting the Specification of System Families, supervised by J.-M. Debaud, J. Bayer, D. Rombach, Diploma Thesis, University of Kaiserslautern, Dec. 1997.

[Mut02] D. Muthig. A Light-weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines. Dissertation im Fachbereich Informatik Universität Kaiserslautern, 2002.

[Nei80] J. Neighbours. Software Construction Using Components, Ph.D. Thesis, Department of Computer Science, University of California, Irvine, 1980.

[Nei89] J. Neighbours. Draco: A Method for Engineering Reusable Software Systems, in Software Reusability - Volume I: Concepts and Models, T. Biggerstaff, A. Perlis (Eds.), ACM Press, Frontier Series, Addison-Wesley, 1989.

[Oes97] B. Oestereich. Objektorientierte Softwareentwicklung mit der unified modeling language. 3. Auflage, Oldenbourg, 1997.

[Oes+03] B. Oestereich, C. Weiss, C. Schröder, T. Weilkiens, A. Lenhard. Objektorientierte Geschäftsprozessmodellierung. dpunkt Verlag 2003.

[Pri90] R. Prieto-Diaz, Domain Analysis: An Introduction, ACM SIGSOFT Software Engineering Notes, vol. 15, p. 47, April 1990.

[PW87] F.P. Prosser, D.E. Winkel. The Art of Digital Design. An Introduction to Top-Down Design. Prentice Hall, 1987.

[Sch97] A.-W. Scheer. Wirtschaftsinformatik, Referenzmodelle für industrielle Geschäftsprozesse. 7. Auflage, Springer 1997.

[Sch00] K. Schmid. Scoping Software Product Lines - An Analysis of an Emerging Technology in [Don00].

Copyright © Fraunhofer IESE 2006

[Sch01] K. Schmid. People Issues in Developing Software Product Lines, in Proceedings of the 2nd Workshop on Software Product Lines: Economics, Architectures, and Implications held in conjunction with the 23rd Internation Conference on Software Engineering (ICSE), 2001.

[Sch+03] A.-W. Scheer, F. Abolhassan, W. Jost, M. Kirchner. Change Management im Unternehmen. Springer 2003.

[SS99] J. Sodhi and P. Sodhi. Software Reuse: Domain Analysis and Design Processes, McGrawHill, 1999.

[TCO00] P. Toft, D. Coleman, and J. Ohta. A Cooperative Model for Cross-Divisional Product Development for a Software Product Line, in[Don00].

[WFM02] Workflow Management Coalition. Workflow Process Definition Interface. – XML Process Definition Language. Document Number WFMC – TC – 1025. Version 1.0, October 2002.

[Wl99] D. M. Weiss and C. T. R. Lai. Software Product Line Engineering: A Family Based Software Engineering Process. Addison-Wesley, 1999.

[WK99] J. B. Warmer and A. G. Kleppe. The Object Constraint Language. Precise Modeling with UML. Addison-Wesley, 1999.

Copyright © Fraunhofer IESE 2006

Document Information

Title:	Principles of Software Product Lines and Process Variants
Date:	February 28, 2004
Report:	IESE-122.06/E
Status:	Final
Distribution:	Public

Copyright 2006, Fraunhofer IESE.
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means including, without limitation, photocopying, recording, or otherwise, without the prior written permission of the publisher. Written permission is not needed if this publication is distributed for non-commercial purposes.