This paper received the Best Paper Award at the
Symposium on Software Reusability '99 (SSR'99),
Los Angeles, May 1999

PuLSE: A Methodology to Develop Software Product Lines

Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen
Fraunhofer Institute for Experimental Software Engineering (IESE)
Sauerwiesen 6
D-67661 Kaiserslautern, Germany
+49 (0) 6301 707 251
{bayer, flege, knauber, laqua, muthig, schmid, widen}@iese.fhg.de

Jean-Marc DeBaud[1]
Lucent Technologies Software Product Line Engineering Laboratories
263 Shuman Boulevard
Naperville, IL 60563, USA
+1 (630) 224 0383
debaud@research.bell-labs.com

ABSTRACT
Software product lines have recently been introduced as one of the most promising advances for efficient software development. Yet upon close examination, there are few guidelines or methodologies available to develop and deploy product lines beyond existing domain engineering approaches. The latter have had mixed success within commercial enterprises because of their deployment complexity, lack of customizability, and especially their misplaced focus, that is on *domains* as opposed to *products*.

To tackle these problems we developed the PuLSE™ (<u>P</u>rod<u>u</u>ct <u>L</u>ine <u>S</u>oftware <u>E</u>ngineering) methodology for the purpose of enabling the conception and deployment of software product lines within a large variety of enterprise contexts. This is achieved via product-centric focus throughout the phases of PuLSE™, customizability of its components, incremental introduction capability, maturity scale for structured evolution, and adaptations to a few main product development situations.

PuLSE™ is the result of a bottom-up effort: the methodology captures and leverages the results (the lessons learned) from our technology transfer activities with our industrial customers. We present in this paper the main ideas behind PuLSE™ and illustrate the methodology with a running example taken from our transfer experience.

Keywords
software product line, domain engineering, domain-specific software architecture

1 Introduction

Problem

Domain engineering has been expected to improve the efficiency of software development because of the notion of economics of scope. Focussing on an area, or domain, where applications significantly overlap enables leveraging the similarities through reuse. Building a reusable infrastructure once for the domain allows multiple applications to be built more efficiently than building them in isolation.

Yet, domain engineering methods have not proved as effective as expected. We believe there are basically three reasons for this: misguided scoping of application area, lack of operational guidance, and overstressed focus on organizational issues.

Domain engineering relies on the notion of an application *domain* to scope the reusable infrastructure. An application domain spans all possible applications in that domain. Domains have proved difficult to scope and engineer from an enterprise stand point because a domain captures many extraneous elements that are of no interest to an enterprise. Hence, the domain view provides little economic basis for scoping decisions. Instead, enterprises focus on particular products (existing, under development, and anticipated). This difference in focus is essential for practically supporting the product-driven needs of enterprises. Products span, as well as, integrate multiple application domains, yet most often only cover a fraction of these whole domains. Therefore, product lines scope based on the economic needs of enterprises.

Existing methods have been either not flexible enough to meet the needs of various industrial situations, or they have been too vague, not applicable without strong additional interpretation and support. A flexible method that can be customized to support various enterprise situations with enough guidance and support is needed.

A lot of work in the literature has focused on the organizational aspect and context for setting up a reusable infrastructure [5,6]. This body of work often takes for granted that the technological problems of how to scope, model and architect the infrastructure have been solved. We do not think this is the case and hence feel strongly that this is the wrong approach. While some general guidelines can be provided, the technology specifics must drive the organization simply because today, the technology is not understood well enough to make it flexible enough to adapt to all sorts of environment contexts.

Context and Approach

The mission of the IESE is to transfer innovative technologies to our customers to help them improve their

1. This work was done while the author was with the IESE.

software engineering and organization practices. Within that context, we have attempted to transition domain engineering know-how. Our initial approach was to use documented methods, such as Commonality Analysis [2], Feature-oriented Domain Analysis [14], or Synthesis [16]. As we used some of their components, problems immediately surfaced. These problems forced us to find solutions throughout the logical phases of the domain engineering life-cycle. Slowly, these solutions together evolved towards an integrated approach of its own: PuLSE (Product Line Software Engineering). PuLSE is the result of a typical bottom-up effort: the methodology captures and leverages the lessons learned from our technology transfer activities with our industrial customers.

This paper is structured as follows. The next section presents a succinct overview of PuLSE and a discussion of related work. Section three presents the PuLSE process in detail using a running example. Section four presents an analysis of PuLSE and the experience we have had using it.

2 PuLSE Overview

Structural Overview

The PuLSE methodology enables the conception and deployment of software product lines within a large variety of enterprise contexts. This is achieved via a strong product-centric focus throughout the phases of PuLSE, the customizability of its components, an incremental introduction capability, a maturity scale for structured enterprise evolution, and adaptations to a few main product development situations.

Figure 1 presents a decomposition of the PuLSE components and phases. PuLSE is articulated around three main elements: the deployment phases, the technical components, and the support components.

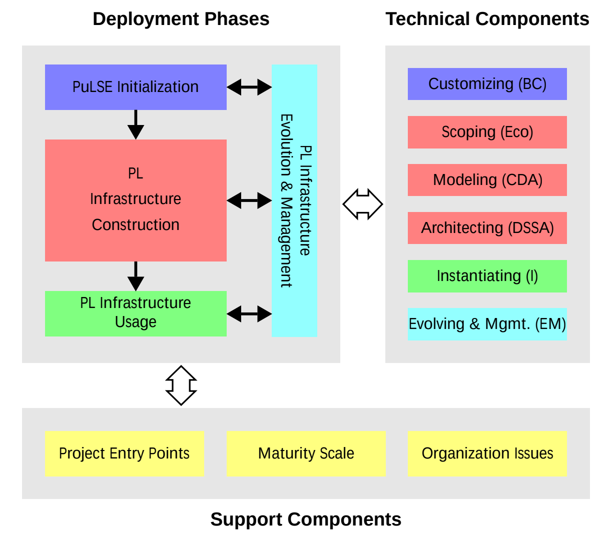

Figure 1. PuLSE Overview

The *deployment phases* are logical stages a product line goes through. They describe the activities performed to set up and use the product line. The phases are:

- Initialization: baseline the enterprise and customize PuLSE as a result
- Infrastructure Construction: scope, model and architect the product line infrastructure
- Infrastructure Usage: use the infrastructure to create product line members
- Evolution and Management: evolve the infrastructure over time and manage it

The *technical components* provide the technical know-how needed to operationalize the product line development. As Figure 1 denotes, they are used throughout the Deployment Phases. A different facet of each is often used in each of the phases – though some components directly correspond to phases. The technical components are:

- Customizing: how to perform the Initialization Phase
- Scoping: how to effectively scope the infrastructure focussing on product definitions
- Modeling: how to model the product characteristics found within the scope of the product line and explicitly denote the product family members
- Architecting: how to develop the reference architecture while maintaining the traceability to the model
- Instantiating: how to perform the Usage Phase
- Evolving and managing: how to integrate misfits in the line, and deal with configuration management issues as products accrue over time

The *support components* are packages of information, or guidelines, which enable a better adaptation, evolution, and deployment of the product line. These components are used by the other elements. They are:

- Project Entry Points: customize PuLSE to major project types. For instance, whether there are legacy assets to be reused or whether multiple projects are independently running and need to be integrated after one realized that much was shareable among them
- Maturity Scale: provide an integration and evolution path for product line adoption to enterprises using PuLSE
- Organization Issues: provide guidelines to set up and maintain the right organization structure for developing and managing product lines

We found it necessary to decompose PuLSE in this manner for a number of reasons, which we motivate historically. First, we developed some of the technical components, in particular Eco, CDA, and DSSA, to tackle the technical difficulties presented by our product emphasis. We quickly realized that a notion of phases was necessary because we faced too much phase overload for most of the components. That is, too many aspects of each technology component were used in different logical stages. This lead to the deployment phase element. There, the contribution and responsibility of each of the technical components is defined while we can preserve the clarity of their technical essence in their canonical definition. Yet, there were additional aspects that had to be taken into consideration, all of them relating to a notion of enterprise context. Hence, the support components were developed.

Related Work

To our knowledge, no product line engineering methods are currently available that are comparable to PuLSE. However, there is much work related to parts of the PuLSE methodology.

Domain engineering methods cover most of the same aspects as PuLSE. However, their focus is different, they lack customizability and they are complex to deploy. Domain engineering methods include Model-Based Software Engineering (MBSE) [14], Organizational Domain Modeling (ODM) [17], Synthesis [16], the Domain-Specific Software Architecture (DSSA) program [18], and the Evolutionary Domain Life-Cycle (EDLC) [7].

Some domain engineering methods include an enterprise/ project baselining step. The related PuLSE component (BC) is grounded in work done on the CMM [11], the Reuse Adoption Guidebook [15], and Experience Factory packages [3]. This work is used in the BC component to support the baselining of an enterprise and customizing, or packaging, an appropriate process for the situation.

PuLSE-Eco is a new approach for defining the economic scope of a product line that draws upon existing work. Conceptually, the closest is the work by Whitey [19]. Also work on economic models for reuse[9], as well as, domain engineering scoping approaches have influenced this component. The major difference of PuLSE-Eco is its emphasis on explicitly grounding the scope definition in business objectives.

Domain analysis, requirements engineering, and knowledge engineering are areas that are related to the PuLSE-CDA component. There are many domain analysis methods, most stemming from the work of Arango and Prieto-Diaz [1]. However, the models are either too specific or they are meant to be tailorable, but lack support. CDA aims to improve upon these methods by providing a flexible domain analysis approach with adequate support. Requirements engineering has also provided input into the modeling workproducts, such as use-cases or state transition diagrams [12].

Work related to PuLSE-DSSA includes the Software Architecture Analysis Method (SAAM) [8]. However, this method focuses only on analysis of existing architectures, while PuLSE-DSSA provides a framework for incrementally creating architectures. There is other work on software architectures that had a strong impact on PuLSE-DSSA [4,13].

3 The PULSE Process

In this section we present PuLSE in detail. This is done by describing the four Deployment Phases. Each phase is described by the relevant Technical Components involved and their interactions within the phase and with other phases. Following the phase descriptions, a section on the Support Components and how they relate to the Deployment Phases is presented.

Each PuLSE phase is illustrated by examples taken from one of the projects in which we started to apply PuLSE. Our partner in this particular project is developing a product line of merchandise information systems. Their primary customer is an enterprise that distributes goods to over 400 supermarkets. Each system within the product line has to support processes related to buying, storing, and selling supermarket goods. Among the main variants are systems for different kinds of points of sale and systems for different kinds of distribution centers (e.g., conventional wholesale stock, cross-docking stock).

Initialization

In the Initialization Phase an instance of the PuLSE methodology is produced that is tailored to the enterprise context in which it will be applied.

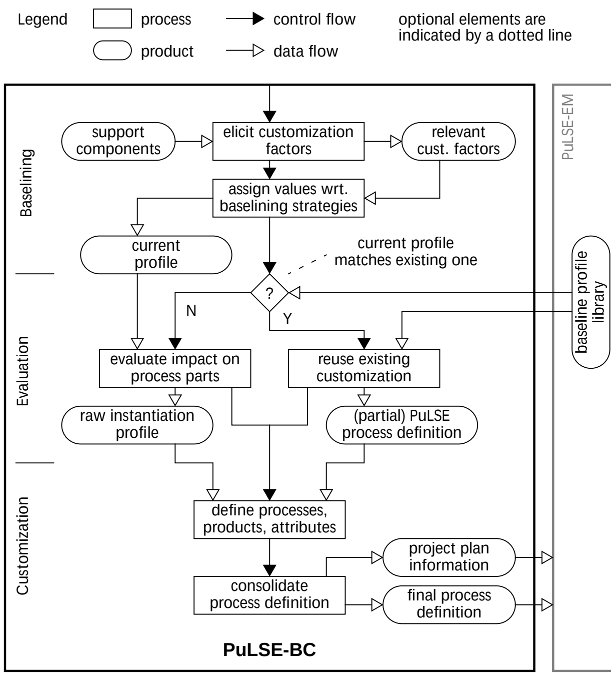
Legend
process
control flow
product
data flow
optional elements are indicated by a dotted line
PuLSE-EM
Baselining
support components
elicit customization factors
relevant cust. factors
assign values wrt. baselining strategies
current profile
current profile matches existing one
?
N
Y
baseline profile library
Evaluation
evaluate impact on process parts
reuse existing customization
raw instantiation profile
(partial) PuLSE process definition
Customization
define processes, products, attributes
project plan information
consolidate process definition
final process definition
PuLSE-BC

Figure 2. PuLSE Initialization Phase

As shown in Figure 2, the Initialization Phase is split into three parts within the Customizing component (PuLSE-BC). These parts are baselining, evaluation, and customization.

During *baselining*, information used for tailoring PuLSE is gathered. The information required is defined by the *customization factors*. These are characteristics of a product line situation (e.g., type of application or amount of resources) that have an impact on the process for developing the product line. Baselining is done by first selecting the relevant customization factors, using the PuLSE *support components*. Then, using the *baselining strategies* of the selected factors, which provide guidelines for gathering the necessary information, values for the selected factors are determined. The *current profile* records the values determined for the enterprise.

The factors, however, are not independent, some factors influence others. A decision tree of factors captures these dependencies and helps to determine the effects on the components of PuLSE. The factors are analyzed in the *evaluation* part. The output of this part is the *raw instantiation profile*, which contains the decisions made for the customization (e.g., the type of workproducts for CDA). When *existing baselining profiles* match the current profile of the enterprise, information from the previous experience can be *reused* for evaluating and also for customizing.

Customization entails deriving a complete process, including the definition of workproducts used, their relations, and their representations, based on the decisions made during evaluation. At this point, information is available that is helpful for *project planning*, such as the expected number of iterations. This information as well as the *final process definition* are passed to PuLSE-EM.

Example: In the merchandise information system example, baselining and evaluation determined that business processes captured as workflows were needed. To support the workflows, business rules, rationales, and a glossary were also identified as required workproducts. This decision is used in customization to tailor the process and notations used during the elicitation of storyboards during domain analysis.

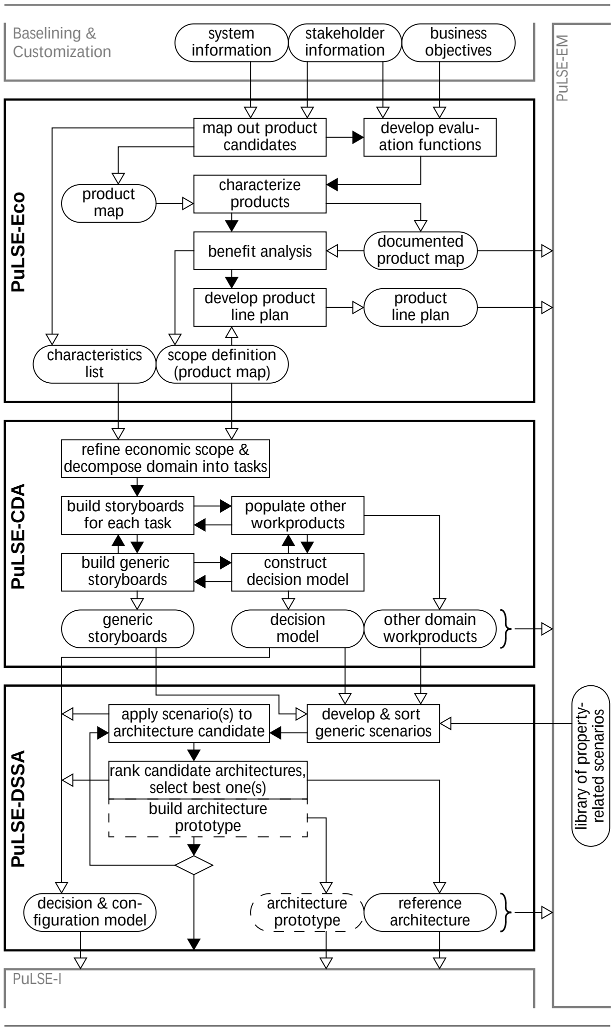
Baselining & Customization
system information
stakeholder information
business objectives
PuLSE-EM
PuLSE-Eco
map out product candidates
develop evaluation functions
product map
characterize products
benefit analysis
documented product map
develop product line plan
product line plan
characteristics list
scope definition (product map)
PuLSE-CDA
refine economic scope & decompose domain into tasks
build storyboards for each task
populate other workproducts
build generic storyboards
construct decision model
generic storyboards
decision model
other domain workproducts
PuLSE-DSSA
apply scenario(s) to architecture candidate
develop & sort generic scenarios
library of property-related scenarios
rank candidate architectures, select best one(s)
build architecture prototype
decision & configuration model
architecture prototype
reference architecture
PuLSE-I

Figure 3. PuLSE Construction Phase

Construction

The purpose of the Construction Phase is to construct the product line infrastructure. This complex task is decomposed into three subtasks, each of them performed by a technical component: PuLSE-Eco helps to determine an economic viable scope for the product line, PuLSE-CDA is used to elicit and articulate product line concepts and their interrelationships, and PuLSE-DSSA is applied to define a software reference architecture for the product line.

Figure 3 presents a high-level overview of the process models of the Construction Phase (i.e., its three technical components). The following subsections explain the process models and illustrate them continuing the running example.

PuLSE-Eco

PuLSE-Eco is used to identify the scope of the product line. The first step is to determine anticipated product line members and map out their characteristics (*map out product candidates*). The anticipated members are used to focus the identification of characteristics and validate them. The product and characteristic information is compiled in the form of a *product map* (see Figure 4). A *characteristics list* is developed that describes the characteristics in detail.

A core idea of PuLSE-Eco is to explicitly base the definition of the product line scope on business objectives that are identified by product line stakeholders. These objectives are insufficient for scoping, they need to be operationalized. Consequently, PuLSE-Eco augments these objectives with evaluation functions (*develop evaluation functions*), which include characterization and benefit functions (see Figure 4). The former evaluate characteristics for each product. The latter use the results of the former to determine the best characteristics and best products for the product line to cover.

In the step *characterize products* the characterization functions are applied on the product/characteristic combinations. The information is added to the product map.

The following step, *benefit analysis*, is the core step of PuLSE-Eco. At this point the gathered information is used to identify the scope. First the values of the characterization functions are used to assign values to the benefit functions. Then the different benefit functions need to be balanced in order to come up with a single scope definition. This is a classical multi-objective decision problem. A large number of techniques for addressing this type of problem have already been described in the literature (e.g., [10]) and can be used directly.

The information about the products, their characteristics, and constraints among them is supplied in the form of the *product line plan* to the PuLSE-EM component.

Example: The table in Figure 4 is a simplified product map for the domain of merchandise information systems. In its rows it shows two top-level subdomains (“Goods Reception” and “Ordering”) with some subtasks. Many rows containing subtasks and further subdomains have been omitted for reasons of space, but the hierarchical structure can be identified. The columns present existing, future, and potential products within the product line. These products are analyzed according to the characterization functions. The rows at the bottom summarize the columns and thus enable an evaluation of the products under certain points of view. The rightmost columns provide an analysis of the significance of the different characteristics. These summaries are determined by the benefit functions. The characteristics in gray table areas have been evaluated to be part of the product line scope.

PuLSE-CDA

In PuLSE-CDA, the product line concepts and their interrelationships are elicited, structured, and documented. This is done using the customized process set up in the Initialization Phase.

PuLSE-CDA initially refines the *economic scope* produced by PuLSE-Eco to specify the boundaries of the product line. The models used (e.g., context diagrams) and the level to which the scope is refined are determined by PuLSE-BC.

Scoping is followed by modeling, in which product line information is elicited and modeled using the workproducts defined by PuLSE-BC. We distinguish between *storyboards* and *other domain workproducts*. Storyboards are used to capture relevant types of action sequences in the domain. These types vary for different domains. Examples are workflow diagrams and message sequence charts. Other workproducts capture additional views on the product line. For example, a data model may be used to capture the data structure common to all systems and varying among them.

Populating the models combines eliciting information about single systems in raw storyboards and other workproducts, and then consolidating this knowledge into generic storyboards and other workproducts that capture variability.

To derive product line member specifications from a product line model, a *decision model* is created that contains a structured set of decisions. Each decision corresponds to a variability in a workproduct together with the set of possible resolutions. To build the specification of a product line member, the decisions are resolved.

The generic storyboards, all additional workproducts, and the decision model are passed to PuLSE-DSSA and PuLSE-EM.

Example: Figure 5 shows a generic storyboard representing the task “Process Goods Reception” that models one of the key areas identified in the product map in Figure 4. The storyboard is captured in DIVERSITY/CDA, a tool we developed to support PuLSE-CDA. Automated support is necessary for managing the large amount of information that exists for industrial applications.

PuLSE-DSSA

PuLSE-DSSA supports the definition of a domain-specific software architecture, which covers current and future applications of the product line as described by a product line model. The basic idea of the process is incremental development of the architecture guided by scenarios. In PuLSE-DSSA scenarios are categorized as either *generic scenarios* (representing functional requirements) or *property-related scenarios* (describing domain-independent quality aspects). The generic scenarios are derived from generic storyboards and the other workproducts created in PuLSE-CDA. Each of them is augmented with a number of property-related scenarios and ranked with respect to architectural importance.

Characterization Functions

s_1: Need to be present

s_2: Competitor products

s_3: Market-share gainer

e_1: Conceptual complexity (1–4)

e_2: Cost estimate (1–4)

X = yes / completely

O = maybe / partially

  = no

Benefit Functions

$$^1: \left\lfloor \frac{\sum_p e_2(c,p)\cdot s_1(c,p)}{p_{max}\cdot e_2(c,p)_{max}} \cdot 10 - 1 \right\rfloor$$

$$^2: \left(\sum_p s_3(c,p)\right) / p_{max}$$

$$^3: \sum_c s_1(c) \vee s_2(c) \vee s_3(c) \qquad (1–8)$$

$$^4: \sum_c s_2(c) \vee s_3(c) \qquad (1–8)$$

$$^5: \left(\sum_{c\in scope} e_2(c)\right) / \left(\sum_c e_2(c)\right) \qquad (1-8)$$

Characteristics \ Products			Existing Applications										Future Applications											Potential Applications						Char.-based		
			Product 1					Product 2					Product 3					Product 4					..	Product 5					...	Effort Saving[1]	Market advantage[2]	...
			s_1	s_2	s_3	e_1	e_2	s_1	s_2	s_3	e_1	e_2	s_1	s_2	s_3	e_1	e_2	s_1	s_2	s_3	e_1	e_2	...	s_1	s_2	s_3	e_1	e_2	...			
Goods Reception	Central storage reception		X	X		3	3	X	X		3	3	X	X		3	2	X	X		2	2		X	X		2	3		8	1	
	Branch local reception		X			2	2	X			2	2		X		1	2			X	2	3		X	X		1	2		5	3	
	Acceptance of non-ordered goods		X	X	O	2	3	X	X	O	2	3	X	X		1	2	X	X	O	2	3		X	X	O	2	3		6	6	
	...																															
Ordering	Reclamation strategy	write reclamation	X	X		2	3	X	X		2	2	X	X		2	2	X	X		3	3		X	X		2	3		7	2	
		charge certain percentage		X		3	2		X		3	4							X	O	3	2			O	O	3	2		6	3	
	Ordering strategy	order at point of sale		X	X	3	3						X			1	2	X	X	O	3	3		X	X		3	3		7	2	
		order at central storage						X	X	0	2	2												X	X		2	1		2	0	
	...																															
...																																

Product-based b.a.		Product 1		Product 2		Product 3		Product 4		...	Product 5		...
	Contribution to domain coverage[3]	5		7		4		7			7		
	Competitive Advantage[4]	3		4		6		3			3		
	Reference Arch. Effort Saving[5]		7		6		6		6			7	
	...												

Figure 4. Example - Simplified Product Map

The architecture development starts with an initial set of generic scenarios that is used to create the initial architecture.

In each of the following iterations, a set of generic scenarios is chosen according to their ranking and then *applied to the current architecture candidate*. The architecture is refined and extended until the reference architecture supports all generic scenarios and has evolved to its final state.

The application of generic scenarios may result in more than one architecture. In this case, the property-related scenarios attached to the currently used generic scenarios are applied to evaluate and *rank the candidate architectures*. The result of this ranking is an architecture candidate (or a small set of candidates) that is further evolved.

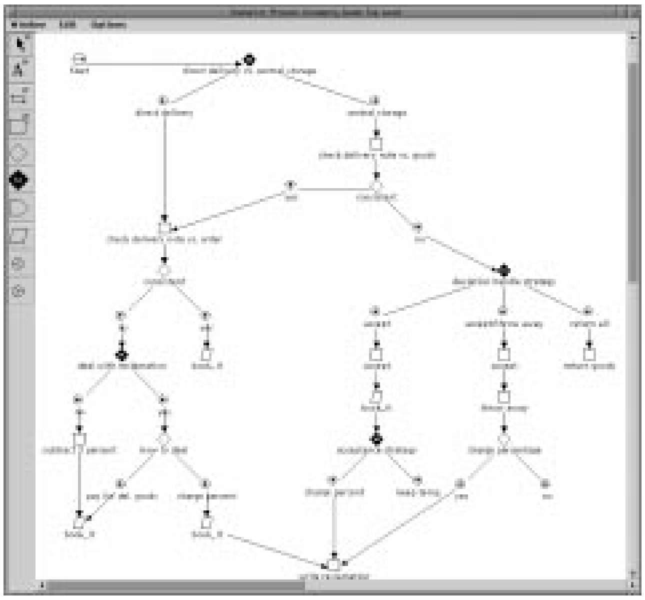
Figure 5. Example - Generic Storyboard

An optional prototype can be built to just support the ranking or it can be created as a structural and evolutionary prototype.

During the creation of the reference architecture, implementation-specific decisions are collected that will have to be resolved during reference instantiation. These decisions and their possible resolutions are captured in the *configuration model* that extends the decision model.

Example: Figure 6 presents a high-level logical view on part of the reference architecture for merchandise information systems. The component “Goods Reception”, two subcomponents of it, and some neighboring components are shown.

Usage

The Usage Phase of PuLSE aims at specifying, instantiating, and validating one member of the product line. This encompasses the instantiation of the product line model and the reference architecture, the creation and/or reuse of products that constitute the instance, and validation of the resulting product. No modifications of the product line (its map, its model, its architecture, or its other assets) are made within the Usage Phase. If changes or extensions to the product line are necessary to build an instance, they are passed as change requests to PuLSE-EM.

Figure 7 shows the process model for the Usage Phase. The initial step in creating a new instance of the product line is to *plan for new product line instance*. In this step, the customer’s requirements are used to plan the development process for the instance. The application of the customer’s requirements to the product map together with the product

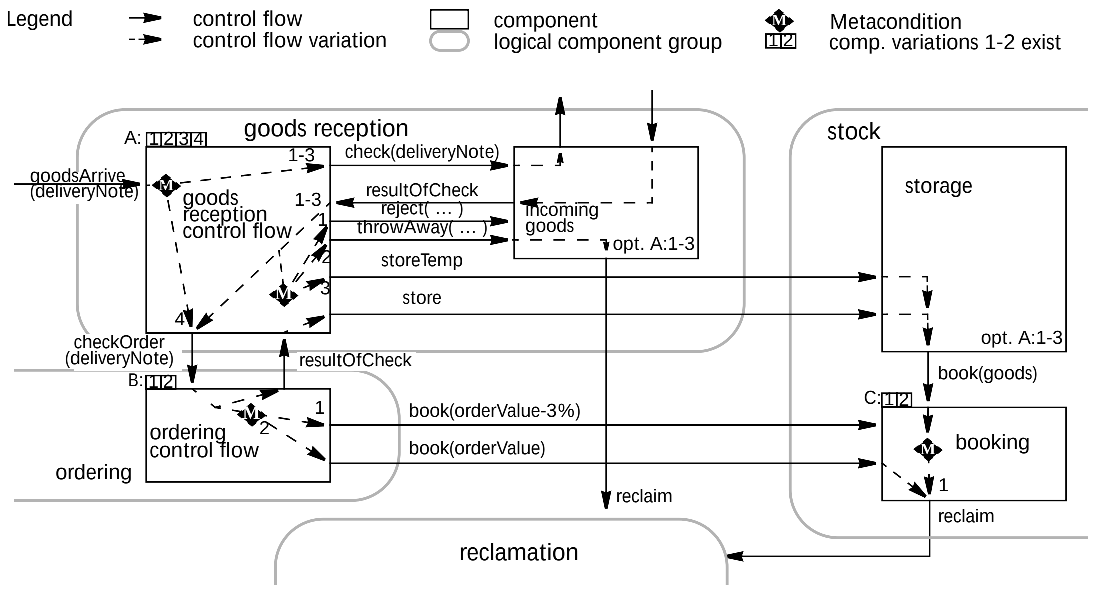

Figure 6. Example - Partial Reference Architecture

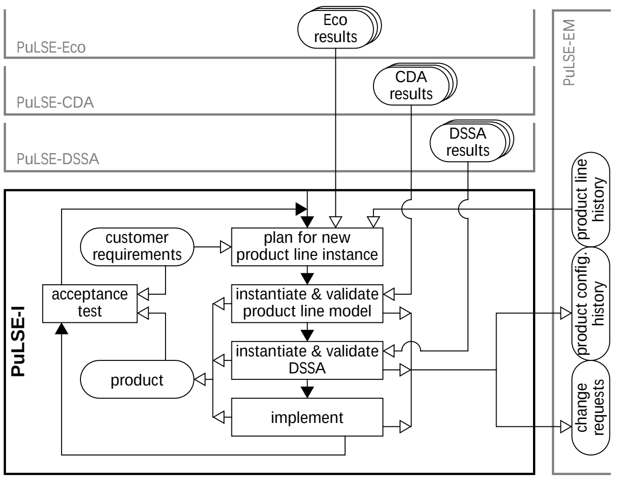

Figure 7. PuLSE Usage Phase

line history enables an estimate about the portion of the new instance that is covered by already existing assets. This estimate is used to set up plans for budget and resources as well as a time table for the instantiation.

In the first instantiation step, the *product line model is instantiated and validated*. Driven by the decision model, the new instance is specified. This specification of the new product is then validated against the customer's requirements. The validated specification is part of the product and is entered into the product configuration history.

Example: Figure 9 shows one specific instance of the generic storyboard in Figure 5 which defines the workflow for a specific product. It is defined using the decision model shown in Figure 8. The particular decision presented there is to support "direct delivery". These are captured in DIVERSITY/CDA.

In the next step, the *DSSA is instantiated and validated*. Driven by the configuration model and the product specification, the architecture for the instance is defined. The architecture is then validated against the product specification. The validated architecture is part of the product and is entered into the product configuration history.

Example: Figure 10 shows an instance of the reference architecture in Figure 6 which supports the workflow defined by the storyboard in Figure 10.

In the following step, the low-level design and the code are developed. Each component stems from one of three sources. The first possibility is that existing product line assets are reused. The second possibility is that assets are created that are within the scope of the product line but have not been designed and implemented yet. The last possibility is that assets are created to fulfill specific requirements of the current instance, and will not be integrated into the product line assets. Requests to change the product line according to these specific requirements have been rejected by PuLSE-EM.

The design and code is then validated against the architecture. The validated code completes the product and is entered into the product configuration history.

Finally, the complete product has to pass an *acceptance test*, which is performed by the customer. This test may cause another iteration of the Usage Phase.

Evolution

The purpose of the PuLSE Evolution Phase is to monitor and control the evolution of the product line infrastructure which is built in the Construction Phase, over time. The Evolution

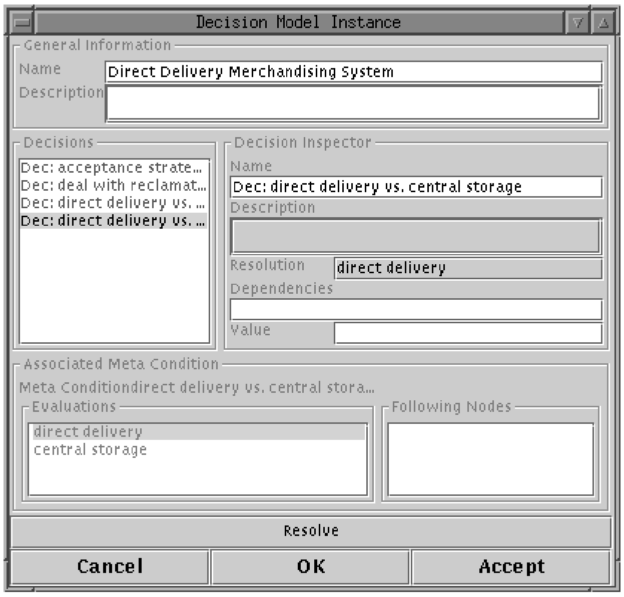
Decision Model Instance
General Information
Name: Direct Delivery Merchandising System
Description:
Decisions
Dec: acceptance strate...
Dec: deal with reclamat...
Dec: direct delivery vs. ...
Dec: direct delivery vs. ...
Decision Inspector
Name: Dec: direct delivery vs. central storage
Description:
Resolution: direct delivery
Dependencies:
Value:
Associated Meta Condition
Meta Conditiondirect delivery vs. central stora...
Evaluations
direct delivery
central storage
Following Nodes

Figure 8. Example - Instance of the Decision Model

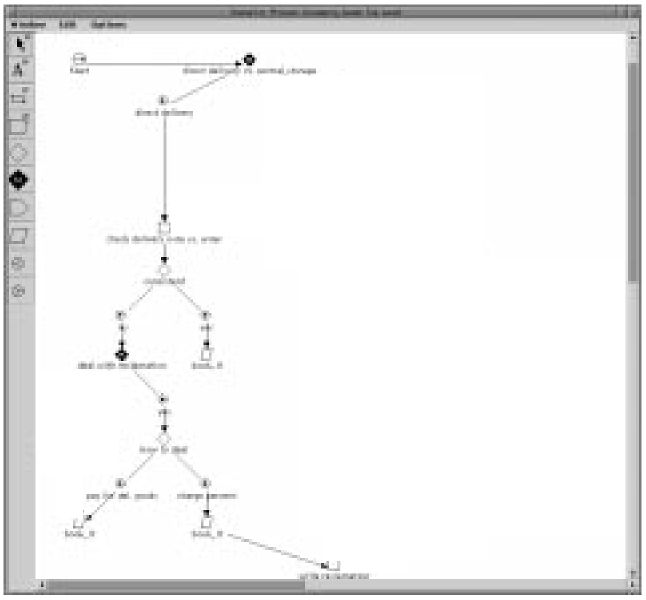

Figure 9. Example - Instantiated Storyboard

Phase applies an instance of the technical component PuLSE-EM (Evolution and Management) customized to the needs of the specific enterprise where PuLSE is applied. Facets of PuLSE-EM also contribute to the Initialization, Construction, and Usage phases (once they have been started).

PuLSE-EM coordinates the activities of the other PuLSE components. It gets various workproducts from the different components, consolidates them, and takes care of their evolution and maintenance. This section highlights the evolutionary aspect of PuLSE-EM. Managing and organizational issues are addressed in section 2 and section .

Figure 11 presents a high-level overview of PuLSE-EM activities and the workproducts involved.

Based on the information provided by PuLSE-BC, PuLSE-EM plans and guides the development of the product line infrastructure.

In all components of the Construction and Usage Phases problems may arise that cause changes to the specifications from previous components. These changes have ramifications, not necessarily limited to the preceeding phase(s) as the following examples demonstrate:

Example: During the application of PuLSE-DSSA problems to adapt to required COTS components may turn up. These problems may affect the scope (handled by PuLSE-Eco), which in turn may cause changes to the generic storyboards or other CDA workproducts.

PuLSE-EM takes care of all these kinds of *change requests*, *consolidates and evaluates* them, *determines* their (potential) ramifications, and *restarts the appropriate steps* of the respective components where they are handled.

Supporting PuLSE

As described in Section 2.1, the support components provide guidelines to help transitioning PuLSE into an enterprise and to lead the enterprise towards a product line development organization.

The PuLSE *project entry points* describe standard situations where PuLSE can be applied. Thus, they help transition PuLSE into a specific enterprise context. Three entry points we already used are:

- Pure PuLSE: It characterizes the situation where a new product line is set up in an enterprise. In that context, all PuLSE components can be established with full traceability among them.
- Evolutionary PuLSE: The different components of PuLSE can be integrated one by one in a currently running development process.
- Reengineering-driven PuLSE: Existing assets can be reused to seed, enrich, or augment the different PuLSE component deliverables.

The PuLSE *maturity scale* is designed to help an enterprise adopt the methodology step by step. It drives towards the usage and integration of the different PuLSE phases and

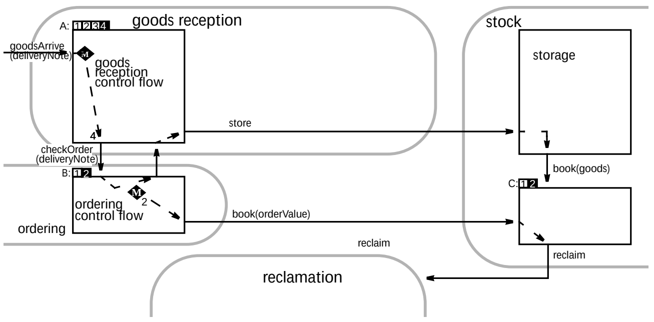

Figure 10. Example - Instantiated DSSA

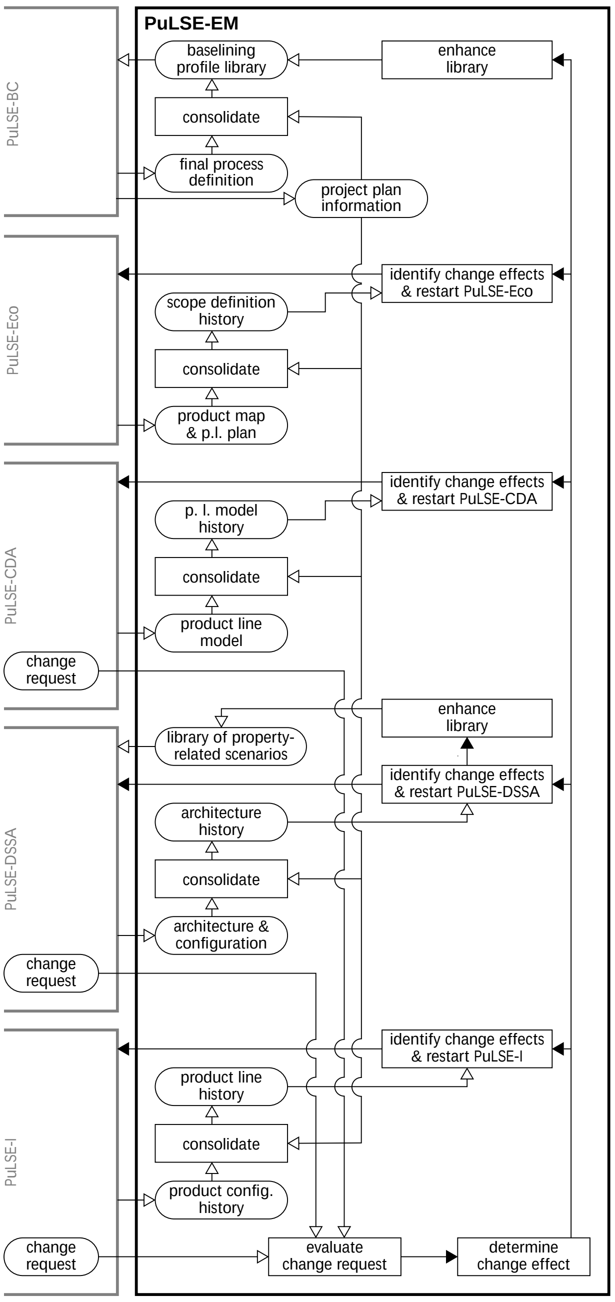

Figure 11. PuLSE Evolution Phase

components so as to help an enterprise ultimately function fully according to a product line mode. Hence, it measures the level of the product line life-cycle adoption within an enterprise. It was inspired by another reuse-driven maturity scale [15]. We have found four levels to be sufficient so far. Each is documented via key elements, which describe fundamental properties that must be exhibited by the current product line process. The levels are:

- Initial: Single PuLSE technology components can be applied independently of one another -- mainly Eco, CDA, DSSA after the necessary customization.
- Full: All components participating to the Construction Phase are used, yet their degree of integration can vary substantially. Partial product instantiation can start.
- Controlled: PuLSE is applied as a complete development life-cycle. Traceability (integration) among the different phases is established and maintained.
- Optimizing: The PuLSE development life-cycle is refined over a number of instances using optimization techniques.

The scale reflects reality as we have experienced it so far. One can rarely expect an enterprise to adopt a complete product line life-cycle immediately from the start. Most would start (have started in our case) small, one component at a time within pilot projects, and then expand the realm of the PuLSE application. Our portfolio today includes four enterprises at the Initial Level, two at the Full Level and one about to reach the Controlled Level (for a few of their large subdomains).

The main PuLSE *organization issue* guidelines span both the development and project organization areas. Both are intertwined. The guidelines are the result of what we have seen work so far. For the development organization, they are:

- Subdivision of the application domain into areas of specialization according to separation of concerns
- Vertical layering decomposition of the development into the application area analysis, architecture, implementation and deployment
- Assignment of permanent personnel to these areas of specialization within the layered decomposition whenever possible
- Supervision and enforcement of the process rules, architecture soundness and evolution

For the project organization the guidelines are:

- Dynamic assignment of personnel to projects according to their responsibility area, that is, developers belong to the development organization, not to the projects.
- Very small project leadership. Project leaders coordinate the project development with the product line.
- A developer assignment ratio of 70% to current project development and 30% on the evolution of the reusable infrastructure seems to be appropriate -- though our data points so far are still limited.

This combination of development and project structure has the following advantages: The personnel identifies with the product line environment, experience reuse is maximized, and there is a minimization of line managers' 'stove-pipe' attitude.

4 Analysis and Experience

As shown in the running example our biggest customer deals with merchandise information systems. We have gained complementary experience from some other cooperations with industrial customers. One of them produces case based reasoning (CBR) tools for various purposes. Targeted towards a complete redesign of their software, we applied PuLSE-CDA to describe the application situations of current and anticipated future products. Based on these we started to apply PuLSE-DSSA to derive a common reference architecture for all of these. As the developed infrastructure seems to be promising PuLSE-Eco was introduced to plan more reliably for approaching new markets, thus PuLSE is applied on the Full Level.

Another customer whose civil engineering software leads the german market in the area of composite construction is on the Full Level too. We use PuLSE there with the final goal of leveraging their currently separate products dealing with different yet overlapping aspects of the same domain into an integrated solution with a CAD frontend. Other domains where we are applying single PuLSE components with industrial partners are stock market data evaluation, CAD systems, human comfort simulations and layout systems.

The experience we have made so far and our progresses are very promising. The following sections analyze some of our most interesting experience:

The contexts of product lines vary greatly. The differences affect how a product line should be established and used (i.e., not all methods will work in all contexts). Therefore, PuLSE-BC uses the context to tailor the other PuLSE components to the needs of the enterprise. However, there is still much human decision support needed to determine the most appropriate fit. To overcome this, we are working on packages of PuLSE customizations that are pretailored to general situations, such as information systems.

PuLSE-Eco provides a good basis for communication and it is easy to use. The only limitation is the size of the product maps: very large maps are hard to handle.

Customizations play a key role in PuLSE-CDA. We have observed that universal models are not appropriate. Domains have different concerns and therefore, require different models to capture their key concepts. PuLSE-CDA is customizable to the relevant domain abstractions and notations. An additional advantage of PuLSE-CDA is the decision model, which aggregates all of the variability from a product line model. This allows for higher-level variability modeling as well as facilitates instantiation of individual systems. Our experience does not yet allow us to provide guidance on customizing the level of detail to which models are captured.

PuLSE-DSSA provides a framework for doing reference architecture development. Even though it can be argued that

PuLSE-DSSA provides only low guidance on the actual design, we claim this one of the major advantages of the component. PuLSE-DSSA does neither require a specific design methodology nor a special architecture description language or technique. It can use whatever is present and established in the enterprise. Another strong point is the clear separation of domain-related and implementation-related decisions; the implementation-related decisions are clearly separated in the configuration model.

The major benefit of PuLSE-I is that it does not require all product line assets to be implemented before the first product can be built. Implementation of assets takes place just as they are needed for a specific product. Therefore using this approach means that there has to be strong supervision to make sure that assets are developed considering the product line context (i.e., not only the current project). PuLSE-I supports this by using the workproducts from the previous Construction Phase and reuses already existing assets systematically.

Our experience shows that PuLSE-EM provides an effective mechanism to propagate change requests that turn up during the Construction and Usage Phases to the responsible components. Even though that approach seems to introduce some overhead and PuLSE-EM needs to be customized carefully to the specific environment, it proves to be a good way to make sure that those components handle the request that are competent for them. Moreover, we expect that when an enterprise applies PuLSE at the Optimizing Level, over time, PuLSE-EM will enable reasoning about the kinds of changes that occur and be able to more effectively evolve the product line infrastructure to future needs.

5 Conclusions

In this paper, we presented PuLSE, a customizable methodology for the conception and deployment of software product lines.

An application of the methodology was shown by presenting an example from our experience. We are currently preparing a technical report on PuLSE, which will describe PuLSE and its application in more detail.

The experience and results of applying PuLSE encourage us to do further research. This involves research in three different directions. First, we want to learn more about the application of PuLSE in practice. Therefore, we try to establish more projects in different domains where we can apply PuLSE. The second direction is process related, where we want to prepare a guidebook that enables enterprises to deploy PuLSE. The last direction is tool support; our aim is to support the complete methodology with an integrated tool suite that encompasses the currently developed domain analysis tool, DIVERSITY/CDA.

References

1. Arango, G. and Prieto-Diaz, R. (eds.) Domain Analysis Concepts and Research Directions. In *Domain Analysis and Software Systems Modeling*, pp. 9-31, IEEE Computer Society Press, 1991.
2. Ardis, M. and Weiss, D. Defining Families: The Commonality Analysis. *Proceedings of the Nineteenth International Conference on Software Engineering*, pp. 649-650, IEEE Computer Society Press, May 1997.
3. Basili, V., Caldiera, G., and Rombach, D. Experience Factory. *Encyclopedia of Software Engineering Volume 1:*469-476, Marciniak, J. ed. John Wiley & Sons, 1994.
4. Bass, L., Clements, P., and Kazman, R. Software Architecture in Practice. Addison-Wesley, 1998
5. Bergey, J. et. al. DoD Product Line Practice Workshop Report. *Technical Report CMU/SEI-98-TR-07*, Carnegie Mellon, May 1998.
6. Foreman, J. Product Line Based Software Development - Significant Results, Future Challenges. *Proceedings of the Software Technology Conference*. April 1996.
7. Gomaa, H., Kerschberg, L., Sugumaran, V., Bosch, C., Tavakoli, I. and O'Hara. L. A knowledge-based software engineering environment for reusable software requirements and architectures. *Automated Software Engineering, 3(3,4)*, pp. 285–307, August 1996.
8. Kazman, R., Abowd, L., Bass, R., and Clements, P. Scenario-Based Analysis of Software Architecture, *IEEE Software*, 11/1996.
9. Lim, W. Reuse economics: A comparison of seventeen models and directions for future research. *Proceedings of the Fourth International Conference on Software Reuse*, pp. 41–50, 1996.
10. Mollaghasemi, M. and Pet-Edwards, J. *Making Multiple-Objective Decisions*. IEEE Computer Society, 1997.
11. Paulk, M., Curtis, B., Chrissis, M., and Weber, C. Capability Maturity Model for Software (Version 1.1). *Technical Report CMU/SEI-93-TR-024*, February 1993.
12. Potts, C., Takahashi, K., and Anton, A. Inquiry-Based Requirments Analysis. *IEEE Software*, pp. 21-32, March 1994
13. Shaw, M. and Garlan, D. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996.
14. Software Engineering Institute, Model-Based Software Engineering. http://www.sei.cmu.edu/technology/mbse/is.html, April 25, 1998.
15. Software Productivity Consortium Services Corporation. Reuse Adoption Guidebook, Version 02.00.05, November 1993.
16. Software Productivity Consortium. Reuse-Driven Software Processes Guidebook, Version 02.00.03. *Technical Report SPC-92019-CMC*, Software Productivity Consortium, November 1993.
17. Software Technology for Adaptable Reliable Systems. Organization Domain Modeling (ODM) Guidebook, Version 2.0. *Unisys STARS Technical Report STARS-VC-A025/001/00*, Reston VA, June 1996.
18. Tracz, W. and Coglianese, L. Domain-Specific Software Architecture Engineering Process Guidelines, *Technical Report ADAGE-IBM-92-02*, Loral Federal Systems, 1992.
19. J. Whitey. Investment analysis of software assets for product lines. *Technical Report CMU/SEI-96-TR010*, Software Engineering Institute, Carnegie Mellon University, 1996.