Fourth Product Line Practice Workshop Report

Len Bass
Paul Clements
Patrick Donohoe
John McGregor
Linda Northrop

February 2000

TECHNICAL REPORT
CMU/SEI-2000-TR-002
ESC-TR-2000-002

blank page (to be thrown out immediately before production)

Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Fourth Product Line Practice Workshop Report

CMU/SEI-2000-TR-002
ESC-TR-2000-002

Len Bass
Paul Clements
Patrick Donohoe
John McGregor
Linda Northrop

February 2000

Product Line Systems Program

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Norton L. Compton, Lt Col., USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Tables

Abstract

The Fourth Software Engineering Institute (SEI) Product Line Practice Workshop was a hands-on meeting held in December 1999 to share industry practices in the area of tool support for software product lines, to explore the technical and non-technical issues involved, and to evolve the SEI Product Line Practice Framework. This report synthesizes the workshop presentations and discussions, which described practices and issues associated with tool support for software product lines.

1 Introduction

1.1 Why Product Line Practice?

An increasing number of organizations are realizing that they can no longer afford to develop multiple software products one product at a time: they are pressured to introduce new products and add functionality to existing ones at a rapid pace. They have explicit needs to achieve large-scale productivity gains, improve time to market, maintain market presence, compensate for an inability to hire, and leverage existing resources. Many organizations are finding that the practice of building sets of related systems together can yield remarkable quantitative improvements in productivity, time to market, product quality, and customer satisfaction. They are adopting a product line approach.

A *product line* is defined to be a group of products sharing a common, managed set of features that satisfy specific needs of a selected market or mission. It is most economical to build a software product line as a product family, where a product family is a group of systems built from a common set of assets.[1] In fact, the products in a software product line can best be leveraged when they share a common architecture that is used to structure components from which the products are built. This common software architecture[2] capitalizes on commonalities in the implementation of the line of products and provides the structural robustness that makes the derivation of software products from software assets economically viable. The architecture and components are central to the set of core assets used to construct and evolve the products in the product line. When we refer to a product line, we always mean a software product line built as a product family.

Some organizations refer to the core asset base that is reused on systems in a product line as a *platform*. Some organizations have other terms for product line, and some organizations differ a bit in their definition of product line. Terminology is not nearly as important to us as the underlying concepts involved, namely, the use of a common asset base in the production of a set of related products.

By product line practice, we mean the systematic use of software assets to assemble, instantiate, generate, or modify the multiple products that constitute a product line. Product line

[1] A software asset is a description of a partial solution (such as a component or design document) or knowledge (such as a requirements database or test procedure) that engineers use to build or modify software products [Withey 96].

[2] A software architecture of a computing system is the structure or structures of the system that consist of software components, the externally visible properties of those components, and the relationships among them [Bass 98a].

practice involves strategic, large-grained reuse as a business enabler. Some organizations have already experienced considerable savings by using a product line approach for software system production. Other organizations are attracted to the idea but are in varying stages of integrating product line practices.

In January 1997, the Software Engineering Institute (SEI) launched a technical initiative, the Product Line Practice Initiative, to help facilitate and accelerate the transition to sound software engineering practices using a product line approach. The goal of this initiative is to provide organizations with an integrated business and technical approach to systematic reuse so that they can more efficiently produce and maintain similar systems of predictable quality at lower cost.

One of the strategies to reach this goal involves direct interaction with and nurturing of the community interested in product line practice. This transition strategy has been executed in part by a series of product line workshops organized by the SEI. The workshop described in this report is the fourth such workshop to bring together international groups of leading practitioners from industry to codify industry-wide best practices in product lines. The results of the previous three workshops are documented in SEI reports [Bass 97, Bass 98b, Bass 99]. The SEI has also refined the workshop results through work with collaboration partners, participation in other workshops, and continued research. In addition, the SEI is producing a framework for product line practice. The SEI's Product Line Practice Framework describes the foundational product line concepts and identifies the essential activities and practices that an organization must master before it can expect to field a product line of software or software-intensive systems successfully. The framework organizes product line practices into practice areas that are categorized according to software engineering, technical management, and organizational management. These categories do not represent job titles, but rather disciplines. The framework is a living document that is evolving as experience with product line practice grows. Version 2 of the framework was made available on the Web in September 1999 [Clements 99].

One of the practice areas yet to be written for the framework is Tool Support. This Fourth Product Line Practice Workshop differed from the previous three workshops in that it focused on exactly one practice area, tool support for product lines.

1.2 About the Workshop

The SEI held the fourth in a series of two-day Product Line Practice Workshops in December 1999 to achieve the following goals:

- Share information and issues about tool support for product lines.
- Stimulate the growth of a network of interest in software product lines.
- Populate the framework with proven practices in the area of tool support.
- Identify gaps where experience is not properly reflected in the framework.

The participants in this workshop were invited based upon our knowledge of their company's experience with strategic software reuse through product lines. The participants were sent a copy of the SEI's Product Line Practice Framework to provide a common focus to structure the workshop presentations and discussions.

The workshop participants included

- Felix Bachmann, SEI
- Len Bass, SEI
- Grady Campbell, SEI
- Paul Clements, SEI
- T.W. Cook, Microelectronics and Computer Technology Consortium (MCC)
- Michel Coriat, Thomson-CSF/LCAT
- Martine Devos, Electronic Data Systems (EDS)
- Patrick Donohoe, SEI
- David Garlan, SEI
- Charles Krueger, BigLever Software, Inc.
- James T. Kurnik, General Motors Corporation, Powertrain
- Robert Marcus, General Motors Corporation
- John McGregor, SEI
- Robert Nord, SEI
- Linda M. Northrop, SEI
- Clifford Stockdill, Raytheon Systems Company
- David M. Weiss, Lucent Technologies

The domains in which the participants have product line experience include telecommunications, television broadcasting, satellite command and control, air traffic control, and automotive engineering. Two of the participants were affiliated with organizations doing research in product line tool support. Each guest was asked to make a presentation explaining his or her organization's experience with software product lines, describing how tools are used to support product lines both in the development and maintenance of core assets and in the development and maintenance of products.

On the second day, participant presentations were summarized. Then the participants divided into three working groups to explore the practices and issues surrounding tool support for product lines from three, albeit somewhat overlapping, angles:

- What is special about tool support for product lines?
- How can tools be used to support product line efforts?
- What variety of tools is needed for product lines?

The working groups then presented their results to the entire group. The workshop concluded with a discussion of how to inform tool vendors about product line needs and how to motivate vendors to satisfy those needs.

1.3 About This Report

This report summarizes the presentations and discussions at the workshop. As such, the report is written primarily for product line champions who are already working or initiating product lines practices in their own organizations. Tool vendors could learn much about the needs of product line organizations and the shortcomings of existing tools. Technical software managers should also benefit from the information.

The report is organized into four main sections that parallel the workshop format:

1. Introduction
2. Product Line Experiences: Highlights of Participants' Presentations
3. Practices and Issues Related to Product Line Tool Support: Working Group Reports
4. Recommendations and Summary

The section following this introduction, "Product Line Experiences: Highlights of Participants' Presentations," synthesizes the product line experience of the workshop participants by providing the highlights of each presentation and the identifying themes that were common to all the presentations. Section 3 is composed of the three working group reports. The recommendations and summary in Section 4 recap major themes and provide suggestions for motivating tool vendors. Additionally, a glossary of terms is provided.

2 Product Line Experiences: Highlights of Participants' Presentations

The workshop began with presentations by the invited participants. This section will briefly summarize the highlights of each participant's presentation and conclude with a discussion of the common themes that emerged.

2.1 Lucent Technologies

The Lucent presentation was centered in the context of reengineering subsystems of a single over-arching mega-system (the 5ESS® telephone switch). The Family-oriented Abstraction, Specification, Translation (FAST) approach [Weiss 99] is used to engineer replacement subsystems efficiently as product families. FAST includes an early and rigorous but straightforward economic analysis to determine if the product line engineering effort will pay off. Inputs to the FAST economic model include the cost of building those members without sharing core assets and the cost of building those members as a product family. The intersection of those two cost curves shows the minimum number of family members that will make the product line investment worthwhile. Not all domains or application areas are good candidates for product lines, and the presentation included "target selection criteria." Domains that make the best candidates for paying off the upfront investment are active (indicated by frequent changes with significant resources required for making those changes), revenue-producing (and thus have the attention of management), and relatively independent.

Domain analysis is an important part of FAST. The scope of the product family emerges as the set of commonalities and variabilities that will be allowed among family members, and these are explicitly documented. In addition to the scope, a FAST domain model also includes common terminology for the family, abstractions that apply to the family, and a mechanism for producing a new system from a statement of that member's place in the family.[3] Constructing this domain model is an explicit step of FAST.

Tool support in place at Lucent to support FAST currently includes a prototype economic modeler, a language development system (to help construct languages for defining a family member and mechanisms for generating software based on such a definition), and a sophisticated process-modeling tool called process and artifact state transition abstraction (PASTA). Word processors are currently used in structured ways to help with the commonality analysis

[3] This is related to the concept of a *production plan* in the SEI Product Line Practice Framework.

exercise that is part of the domain analysis step; requirements for a prototype special-purpose tool have been written. The tool used for architecture representation is, alas, PowerPoint™.

2.2 Electronic Data Systems (EDS)

Our participant from EDS had worked on several product lines in other organizations before her current assignment, and she shared her overall knowledge with the group. She articulated a long list of clearly defined product line risk conditions learned from hard experience. These included tackling a scope that is too large, management expecting a return on investment too soon, a core asset development effort without its own budget and without its own team, not producing an explicit domain model, and taking inappropriate measurements (or none at all) to help track progress. She stressed the power of reflective software—software that reacts to its own state and configuration—for sophisticated product line capabilities. Under this scheme, those aspects of the business model and applications that must be easy to change are captured and stored in a central repository. Then in lieu of hard-wired applications, a small set of operational tools interpret those specifications at runtime and adapt themselves dynamically to the variation called for. She emphasized the use of code tagging as a straightforward, economical way to embed necessary information within a core asset. Code tags are essentially comments that can be read and acted on by various tools; compile-time macros are one example of code tags that cause the compile to produce different versions of code depending on the value of certain parameters. Code tags can also be embedded to interact with configuration management systems, provide traceability to requirements, instantiate architectural variation points, and so forth.

In her world, core assets include business rules, object models (application frameworks), authorization procedures, standard applications, layouts, workflow processes, and stored queries. Tools to manage these assets include editors, browsers, and viewers of appropriate types.

The result is a *de facto* language that is automatically processed to generate the appropriate runtime behavior of a desired product. In many ways, this is a variation of the FAST approach mentioned earlier, in which an application-specific language is designed to carry the commonalities and express the variabilities, and then software is written to process the language that produces the desired family members. Our participant referred to this approach as "building softer software."

2.3 General Motors Corporation, Powertrain

General Motors Powertrain (GMPT) builds embedded controller software for General Motors engine and transmission controllers. This product line is densely populated; many variations are produced. To keep control over the situation, strong reliance on a common architecture is stressed. Software is composed of "rings," where a ring is a functional grouping of software. Rings come in "flavors" that reflect variation in functionality, quality attributes such as performance, and communication mechanisms employed. There are approximately 150 rings in the core Powertrain software. Each ring is composed of a subset of seven sub-classifications.

Examples of the sub-classifications include “system diagnostics,” “on-vehicle communication,” and “off-vehicle communication.” Every product build uses the same rings, and all rings provide the same interfaces, but different flavors are used to configure the software for a particular application.

This well-defined architecture makes building an application a straightforward process of checking out the right components, tailoring them in well-defined ways, and performing integration. This in turn leads to an emphasis on process, and the necessity of proper “bookkeeping” to keep all of the components, flavors, and variations straight.

Toward this end, GMPT uses a standard off-the-shelf configuration management system to perform version control over their core assets. Rings are maintained as the central (software) core assets; all flavors of a ring are under the control of a single ring project and thus may always be found in a single place. (Other core assets are compilation scripts to produce a running application out of the selected ring instances.)

GMPT also uses an off-the-shelf database engine that has been customized to store a ring’s interface contracts and features, to help manage algorithm and software maintenance, to help gage the system impact of adding new features, to track the “uses”/“is used by” relation among the rings to manage interfaces, to track which rings are used in which products, and to help build a project by managing the list of features that the build (i.e., the new product) will support. Interestingly, this last capability is allowing GMPT to move from a component-based view of system building to a feature-based view that is more in tune with customers’ needs.

2.4 Raytheon Systems Company

Raytheon is building a product line of satellite ground control systems called Eclipse. Eclipse is based on an earlier product line effort of such systems for the U.S. government. A satellite ground control system must accept telemetry and other information from the satellite, and it must prepare data (commands, principally) for transmission up to the satellite at predetermined times. Variation in the product line comes from the variation in the satellites with which systems must interact: number (a single satellite vs. a constellation); satellites in different orbits (e.g., geosynchronous vs. low-earth); different stabilization methods (e.g., spin-stabilized); different geometries; different command sets, formats, and rates; and a host of other concerns.

Raytheon presented a clear picture of a large commercial product line effort using a suite of the best available commercial tools to help them manage their production. TD Technologies’ SLATE™ tool handles their requirements management task. This includes requirements text capture, requirements tracing, test plan tracing, and tracing to implementation components. They develop and maintain their product line architecture (and product architectures derived from it) with Rational’s Rose™, using use cases, the static (logical) view, and sequence diagrams. ClearCase® handles version control of all the project’s source code, developers’ proj-

ect files, software test drivers, and documentation. Problem tracking is performed with DDTS™, and is integrated with ClearCase® to map bug fixes with versions.

The Raytheon participant was able to share a wealth of practical experience with using many of the current tools that are available to support (or, more likely, pressed into service to support) product lines. For instance:

- A simple word processor would have been easier to employ for use case capture, although obviously a word processor will not produce anything that can interact with other views of the architecture captured by the architecture tool.
- The project is unable to specify project parameters (for building an instance product) in a single location because the tool environment is distributed and not well integrated.
- Some tools effectively preclude two versions of a product existing on the same workstation at the same time.

He noted that they use the Microsoft® Windows NT® development environment and that choice leads to some of these problems. Specifically, the inability to specify project parameters in one place is related to limitations in Microsoft VisualStudio, and the inability to install multiple versions of a product on a workstation is related to how products are installed on Windows [sprinkling dynamic link libraries (DLLs) all over, meddling with the registry, etc.].

2.5 Thomson CSF/LSAT

Our Thomson participant painted a clear picture of a large industrial product line employing a suite of the current best-of-breed commercially available tools. Firmly embracing the dual life-cycle model for product lines (domain engineering to produce core assets, application engineering to produce products), Thomson divides their tool needs into the following categories: managing requirements, architecture, components, process, repository, traceability, and documentation. Through an effective series of “what we want/what we get” viewgraphs, he communicated the nature and extent of the gulf between current tool support and what would be desired to truly support product line capabilities. Table 1 shows highlights.

Another area of interest was “derivation management,” which is the ability to bind variation points and produce tailored versions of assets to work in a product. At best, derivation management will produce products in an automated or semi-automated fashion. This requires broad cross-tool communication, if not integration, and takes a holistic view of the entire process. For this need, Thomson has resorted to hand-written scripts that work for the various other tools they employ. Derivation management marks the largest open issue for product line tool support.

“Buy the best and do the rest” was the advice given by the Thomson participant, but as one might imagine, the lack of tool integration was high on the list of sore points. Thomson has addressed this need somewhat by trying to partner with local tool vendors where possible, availing themselves of on-site help and even using creative hiring and trading practices to bring in tool experts to help sew up some of the many seams among the tools. Finally, “buy

the best" comes at a price; in Thomson's case, this price is in the neighborhood of $20,000 per workstation.

Area	What we want	What we get
Requirements	• Product line software system requirements tool • Representation of system context (to help with scoping) • Parameterization of textual requirements • Analysis model with variabilities	• Software system requirements tool: QSS DOORS® • Lack of user-friendly requirements capture capability • Limited analysis capability that includes variations
Architecture	• Product line software system architecture tool • Rich set of views modeled • Models including variation points	• Software system architecture tool: Rose™ • Limited number of views • UML™ not fully supported • Variations points not well supported
Software Components	• Component design, specification, implementation with variation points	• Rational Rose™, plus CORBA™-compliant framework • Only skeleton generation with UML™ tools
Process	• Define activity hierarchy (for domain engineering and application engineering) • Stakeholder to-do lists	• Rational ClearGuide™ • Too tightly aligned with ClearCase®
Repository	• Ability to re-create any version of any product • Ability to compare versions (diff)	• Rational ClearCase® • File rather than object versioning • No support for publish/subscribe for assets
Traceability	• Information trace between assets • Information trace within assets • Inter/intra-asset relationships • Navigation between assets	• DOORS® for inter-asset traceability only • Rose™ helps with assets in its realm • Limits on tool importation capabilities
Documentation	• Standard documents • Documentation of assets • Integration with other tools	• DocEXPRESS®/Word • Integrate with DOORS®/Rose™ products • Templates for output documents

Table 1: Comparison of Needed Product Line Tools Versus Available Product Line Tools

2.6 General Motors Corporation

The representative from General Motors (GM) shared with us his company's approach to exploiting reusable assets across a wide variety of application areas. The approach was centered on what he called an architecture delivery process (ADP) that is used by projects to define and implement all deliverables relating to the architecture of the applications. The ADP is viewed as a complement to the system delivery process that has enjoyed long acceptance at GM and is primarily responsible for the delivery of working products. The ADP helps bridge the gap between the repository of reusable assets and delivered systems by making developers pay attention to the long-term enterprise architecture goals. At GM, reusable assets are

grouped into business domain models (business process and system context), application frameworks and object models, product standards, and existing infrastructure. For example, specific business domain model assets include workflow diagrams, functional task descriptions, business object descriptions, system context diagrams, scenarios, and use cases. These categories organize a work breakdown structure for the ADP, and the asset repositories are indexed by this work breakdown structure for easy access and tracking.

Tools are in place to manage these assets as follows:

- business process modeling: ProVision
- architecture delivery tools: Visio®
- data modeling: ERwin™, Rational Unified Modeling Language (UML™), Excel, and Visio®
- metadata, data warehousing: Platinum Repository™ on mainframe
- process description and reusable asset repository: CAProcess Engineer
- configuration management: ClearCase® and Continuus/Configuration Management (CM™)

In this participant's opinion, there still exists a clear need for a process to bridge reuse repositories and application-building projects, and a tool is needed to support the process that (among other things) will deliver reusable assets at appropriate times.

2.7 MCC

The Microelectronics and Computer Consortium Corporation (MCC) is a consortium of the world's leading computer, semiconductor, and electronics manufacturers, and users and producers of information technology. MCC members define and participate in highly leveraged research and development projects that meet their strategic business requirements for electronic systems and information technology. MCC members are eligible to participate in MCC projects and fund only the projects of most interest to them. One such research and development project is called Software & Systems Engineering Productivity (SSEP), whose participants have come together to design and produce a tool environment for building software product lines.

SSEP has taken an approach based on the current work in the field of architecture description languages (ADLs). ADLs provide for the creation, analysis, and evolution of software and system architectures, and also, to varying extents, the generation of software systems that are compliant with those architectures. One such ADL is called ACME. Developed at Carnegie Mellon University, ACME was designed to be an interchange language for use in translating between other ADLs. Such translation would, among other things, allow an architect to quickly apply the analytical capabilities native to each ADL into which the architecture could be translated. SSEP has implemented the ACME language in extensible markup language (XML) and augmented it with an integrated suite of ancillary tools, including

- a repository manager
- a visual editor for the architecture language
- a link manager to capture and navigate the links between product line artifacts and to serve as a component selector
- a dependency checker to check the completeness of an instance architecture
- a composition tool to create an application using an instance architecture and an asset base
- a tool (based on the University of Southern California's WinWin tool) to capture requirements, rationale, and variability
- a scenario manager to capture scenarios for use in domain engineering

These tools are in various states of readiness, from "completed" to "prototyped." Product line requirements will be captured using the requirements tool and the scenario manager. Based on the requirements, an architecture can be sketched out and analysis performed as necessary by translating the architecture into other ADLs and exploiting their native analytical capabilities. Infrastructure and components can then be specified, with variability mechanisms built in, to populate the repository. At each step, rationale is captured and dependency links are maintained. To build a product, specific requirements and/or scenarios are chosen, and the links are used to extract components and bind variabilities.

2.8 BigLever Software, Inc.

The final presentation was, like that of MCC, from the perspective of a producer rather than a consumer of product line technology. This participant represented a small start-up company whose mission is to provide tools and infrastructures for "low-overhead product line architectures." While the company is still in its embryonic stages, it is instructive to learn about the tools that are being given high priority. These include

- an editor for architectural parameters
- an infrastructure for managing common and variant core assets based on architectural parameters
- an instantiation engine for creating products from the common and variant core assets based on architectural parameters
- a tool for semantic analysis
- a tool for impact analysis
- a "multi-dimensional configuration management" tool
- a tool to search for and manage commonality among assets, to aid in the identification and combination of almost-alike variations

Taken as a set, these tools are intended to prevent products from varying from the product line architecture across the entire product line life cycle. For example, the tools will prohibit a

product developer from modifying a core asset for use in a single product. The tool suite thus reflects a strong sense of product line process.

2.9 Common Themes

What can we conclude from our participants' presentations? First, we must conclude that there is obviously no completely suitable tool or tools for engineering a software product line. Nor does there seem to be a proposal for one, although our two tool-builder participants have strong ideas about some of the pieces that are necessary in an all-encompassing environment.

Not surprisingly, given that we must make do with a menagerie of loosely connected tools, interoperability emerges as the greatest visible need. Making tools interoperate is critical, and hard. Interoperability is most important for maintaining traceability and dependency links among artifacts of different types (for example, from a requirement to an architecture variation point to a specific version of a specific component, to a test case). One participant summed up the need by lamenting that he was unable to specify the parameters for one application in a single place. It follows, then, that analysis of these project parameters (for completeness and consistency) cannot be performed by any single tool, if at all.

Besides not talking to each other, the tools that we have today suffer from singular deficiencies such as the following:

- They are expensive.
- They often fail to share information among simultaneous or distributed users.
- They don't provide quite the right information.
- They are hard to learn.

Training is a major consideration, and there is a clear notion of familiarity with a tool versus mastery of a tool. If the tools were easy to use, familiarity would be sufficient. Our participants report that familiarity is clearly not sufficient and that "tool wizard" is a recognized (and necessary) role in their projects.

There are also areas of product line engineering where there is little tool support at all. For example, there is almost nothing available to help with the problem of setting the product line's scope, or for managing it so that a new candidate product can be analyzed to see if it is in or out of scope. Support for generation and/or composition of the products from the core assets is also meager at best. Evolving the product line as a whole is now handled manually or piecemeal.

A clear theme that emerged from our participants' presentations was the importance of establishing, maintaining, and adhering to a clear process while engineering a product line. Process applies to domain engineering (the establishment of the core assets) as well as to applica-

tion engineering (the construction of products from the core assets). The process should drive the selection and use of tools, and not the other way around, and the tools should support the chosen process. For example, the tools should disallow divergence, which comes about when product builders check out core assets and modify them at will to fit their product without regard for the rest of the product line. Tools should also help with architectural enforcement, making sure that products adhere to the product line architecture (or to allowed variations of it). Towards this end, there is a clear need for tools that help with the creation, implementation, modeling, and evolution of processes for product line engineering.

Two difficult questions emerged as common themes of the presentations. First was a question that any product line tool environment should endeavor to answer: “How do I find the information I need right now?” And second, “How would we get a vendor to build the ideal, integrated tool suite for product lines even if we could define it?”

These and other questions were discussed in the working groups.

3 Practices and Issues Related to Product Line Tool Support: Working Group Reports

3.1 What Is Special About Tool Support for Product Lines?

One working group explored answers to the question: "What is special about tool support for product lines?" This question was addressed in three steps:

- How does development in a product line environment differ from a conventional environment?
- What do these differences imply about criteria for selecting tools to support product line-based development?
- What are the risks associated with tools in a product line effort?

Finally, a tools "wish list" was constructed that arose naturally from the discussion about the product line development environment.

3.1.1 Differences in Product Line Development

There are a number of differences between development in a product line environment and in a conventional environment.

Artifacts are long lived. The lifetime of an individual artifact[4] is extended because it continues to be reused across a number of products. Development techniques also allow a portion of an artifact to be reused as opposed to an all-or-nothing approach. Relationships are established between artifacts and the product line architecture, between the artifacts and specific products, and between two or more artifacts.

There are more and different types of artifacts. The need to communicate across numerous product teams results in more artifacts being created. Certain types of models, such as domain models, that might be informal in single product production are formalized in a product line environment. New types of artifacts such as reuse guides are created for product lines.

[4] Artifacts correspond to what are called *core assets* in the SEI Product Line Practice Framework.

Artifacts are more complex. In a product line environment, artifacts often must support the variability inherent among the various products in the product line. The artifact may describe a complete set of alternatives or it may describe a set of criteria for determining the suitability of a substitute.

Architecture is created by a small group but used by a large group. Product line development is architecture driven. A dedicated group shapes the product line architecture that is then used, with modifications, by each product team. This requires a precise and expressive representation and tools that can capture the representation either from an original description or from an existing set of artifacts.

Different economic questions are asked. Product line teams can and do conduct more detailed economic studies. The justification for including a specific point of variation can be tied to specific products that can be produced because this variation point is incorporated into the architecture. A business case for these specific products might be warranted to justify the added complexity inflicted on the core assets by addition of the variation point.

Project management is multidimensional. A project manager is constrained not only by the limits on the specific product project but by decisions made by product line personnel. Information flows from the product line team to individual product teams, and it flows in the reverse direction as well. Artifacts may also be shared between product teams independent of the product line.

Configuration management is multidimensional. A product line effort produces assets that are reused across different products, and each product has multiple versions. The configuration management process must be able to reconstruct any given build of any product by assembling a variety of assets including architectural designs, analysis models, and requirements.

3.1.2 Criteria for Selecting Tools

Given these differences between a product line approach and a single product development effort, there is an impact on the tools that are used in development. This impact affects tools for

1. configuration management
2. requirements discovery and management
3. architecture modeling
4. project management
5. reverse engineering
6. impact analysis/traceability
7. regression testing

8. economic modeling
9. design modeling
10. build management

Available tools in these categories, as they are currently produced, are not adequate for a product line development effort. The following list of criteria describes the properties needed in tools for product line development efforts. [An asterisk (*) indicates that the criterion applies to all of the tools 1-10 listed above; otherwise, a set of reference numbers indicate the specific affected tools by number.] This list should be especially enlightening to tool vendors.

- **Interoperability** - The artifacts in a product line environment are long lived and need to pass through many of the development phases. The tools should be able to handle output from other tools and produce input for other tools. (*)
- **Ability to handle new types of artifacts** - The product line environment formalizes abstract products such as specifications, models, and plans. To be applicable, tools must be able to handle this wider array of artifacts. (*)
- **Ability to support traceability** - There should be a linkage between artifacts so that the impact of a change in one artifact can quickly and easily be propagated to related artifacts. (*, but particularly 6)
- **Requirements management** - There are two levels of requirements in a product line effort: requirements applicable to all products in the product line and those that apply specifically to a particular product. The requirements process must be supported by a tool that can recognize this distinction but can maintain all of the requirements from all levels. In other words, the tool must be able to handle variation points effectively. (2)
- **Ability to capture future requirements** - The requirements process should be able to distinguish current requirements that must be met from future requirements that should be considered during design but are not mandatory. (1, 2, 3, 6, 8)
- **Validation** - Tools must support the validation of every artifact. The ability to “diff” artifacts and to compare to standards is helpful in accomplishing the validation. (5, 6, 7)
- **Tools tailorable to processes** - Tools often embody portions of a development process within their procedures. Tools need to be modifiable so that these process pieces can be tailored to follow the effort’s defined process. (* except for 5)
- **Vendor stability** - Due to the long lived nature of product line efforts and artifacts, vendors and their tools should be available for a sufficiently long period to prevent churn in product line artifacts. (*)
- **Flexible licensing** - Licenses for tools should allow floating licenses that can easily be transferred from one product team to another. (1, 7, 9, 10)
- **Conformance to standards** - All tools should conform to and support existing standards. For example, design tools should enforce correct UML™ syntax. (*)
- **Enforcement in creation** - As new artifacts are created, it should be possible to enforce constraints such as permissible variability and mandatory commonality. (1, 2, 3, 9, 10)
- **Scalability** - Product line efforts produce a large number of artifacts that must be maintained for a long period of time. Many of these artifacts are larger than normal because they must accommodate the variation across products. In particular, design models tend

to be much larger than normal. The common code base also will be larger. Design tools and tools that manage the configuration of the products must be able to handle larger than normal sets of artifacts without performance degradation. (1, 9)

- **Ability to handle complexity** - In addition to producing a large number of artifacts, product line efforts impose a complex set of relations among the artifacts. Tools should be able to accommodate the added complexity. (*)
- **Ability to tag variation points** - The ability to recognize variation points as specific entities is a useful attribute for product line tools. (*)
- **Ability to support multidimensionality of project management** - The management environment in a product line effort is "multidimensional" in the sense that the product line and multiple products form one set of levels within which there are multiple versions. Development schedules must coordinate releases of the product line artifacts to product teams, releases of products to customers, and releases from external vendors into the development environment. (4)
- **Version architectures** - An effective development process includes the ability to reproduce any build, version, or release of a product. This includes not only the code but also the documentation, detailed design, and architectures. Tools must be able to support multiple versions of the product line architecture and the individual product architectures as part of re-creating a specific product version. (1, 3)
- **Architecture as an asset** - The product line architecture is used to accelerate the process of developing architectures for specific products. Tools should support the representation of variations and commonality among product architectures. The architectures should be maintained and annotated just like any attribute. (1, 3)
- **Represent rationale** - Design decisions that are not documented will be debated many times over. It should be possible to attach annotations to places in the design diagrams, code, and other documents that capture the reasons for the decision and which alternatives were considered. (2, 3, 4, 6, 9, 10)

3.1.3 Risks

The fact that we have to make do with single-system tools, and often not very good ones at that, leads to the following risks:

- Resources may be expended unnecessarily due to the lack of tool support. If needed tools are not available, either the work will be done manually, taking longer than necessary, or the work will not be done at all, resulting in missing or inadequate artifacts. A specific example considered by the group is the inability to "diff" artifacts due to their complex structure. Given the current state of tools, the probability of this risk becoming a problem is high. The resulting loss will be time/money or missing artifacts so that the anticipated productivity gains are not fully realized.
- Artifacts may become inconsistent due to missing linkages between artifacts. If tools do not interoperate and provide traceability between artifacts, changes made to one artifact may not be noticed by owners of other related attributes. Eventually the artifacts become so inconsistent that it may no longer be clear what information is correct. Given the current state of tools, the probability of this risk becoming a problem is high. The resulting loss is the time required to resolve the uncertainty of which elements of which artifacts are the most accurate. An additional loss may be the time to re-create information that is lost due to assumptions about incorrect artifacts.

- Tools may need to be replaced due to the change to a product line approach. Single product development projects may use an informal suite of tools including freeware and shareware. Product line development efforts will require more robust tools with a larger capacity. Given the current state of tools, the probability of this risk becoming a problem is moderate since few projects currently have extensive tool sets. The resulting loss is the amount of resources to purchase the large tool set and to train project personnel in their use.
- The project may not achieve the expected results due to an overreliance on tools without sufficient oversight. The probability of this risk becoming a problem is moderate. Most developers are well aware of the limitations of tools, although many managers may be tempted to view problems as solved if a tool has been purchased to address the problem. The resulting loss may be oversimplified artifacts.
- The tool set may require unanticipated resources due to a reliance on local personnel for extensive local modification and maintenance of available tools. The local tools group may need to be expanded if the product line effort attempts to use inadequate tools to support its development process. The probability of this risk becoming a problem relates directly to the skill of the local tools group. It may be relatively low if there are skilled people using high-quality tools. The resulting loss will be time/money spent upgrading the tools group.
- The tool set may become difficult to maintain due to the large amount of local modification. As new versions of tools are released by vendors, the local modifications must be ported from the previous versions. This can become an unacceptably large responsibility. Given the current state of tools, the probability of this risk becoming a problem is high. The resulting loss will be wasted resources, as repetitive application of the same fixes require the tools group's time.

3.1.4 Tools "Wish List"

Given the differences between a product line development effort and a single product effort and the criteria that tools must meet for product lines, a product line development team needs support from the following types of tools. Instances of some of these tools exist in research or prototype form; however, most fail to be sufficiently industrial strength for practical use. Ideally, there should be tools ready for practical use to support product line efforts in the following areas:

- **Architecture reengineering/extraction** - It should be possible to synchronize the architecture and code, even in projects that did not do an architectural design before coding. There should be a tool that supports architecture reconstruction and that fits into an iterative process so that an architecture can be reconstructed for applications constructed by assembling components.
- **Testing**
  - **Conformance** - It should be possible to apply a set of tests that evaluate the degree to which a design or system conforms to the constraints expressed in the product line assets. For example, the architecture extraction tool might be used on product code to determine whether its structure conforms to the specified architecture and whether or not that architecture conforms to the product line architecture.

  - **Regression** - There should be a tool that can maintain a large set of tests that periodically can be applied to the inventory of reusable assets. The tool should interoperate with the linkage tool so that regression testing is triggered according to a specified set of conditions.

- **Modeling tools**
  - **Domain** - The set of domain models for a product line will be large and interconnected. These models provide a working vocabulary for the project and a set of structural relationships. A tool is needed to support the development of appropriate models and must interoperate with the linkage tool and the architecture tool to relate the domain concepts to architectural structures.
  - **Process** - The variation in products and teams that build them leads to a potential variation in process definitions used in the individual product development activities. A process modeling tool should allow new process definitions to be quickly derived from existing process definitions according to specific variation points.
  - **Economic justification** - There should be a tool that supports the computation of revenues and costs for the overall product line and also supports "what-if" queries to support proposals for individual products.
- **Product line scoping** - There should be a product line scoping tool that interoperates with the linkage tool and the modeling tools. The tool would support exploration of both domain models and requirements models to develop criteria for determining whether a specific product should be incorporated in the product line or not.
- **Configuration management**

  The configuration management system should be able to re-create any artifact at certain specified intervals in time.
  - **Version control** - The version control mechanism must operate along an indeterminable number of dimensions. For example, a product has new releases with expanded and improved functionality over time. It also can be the root of a set of variants that address very closely related, but slightly different, requirements sets.
  - **"diff"ing non-text artifacts** - A structural differencing tool is needed that can compare complex entities and provide understandable analyses of their differences. For example, the original architecture and the reverse-engineered architecture could be compared to determine whether the code faithfully conforms to the original architecture.
- **Visualization** - It would be useful to have a visualization tool that interoperates with the linkage tool to make relationships visible. This tool should, in turn, allow the modeling tool to be initiated from the visualization tool so that artifacts and relationships are modified.
- **Linkage tool** - Any artifact may be related to any other. A tool that supports linking one artifact to another and annotating any artifact would allow project personnel to express any relationship that might be useful. This tool must provide a transparent layer between the set of artifacts and all of the other tools. This allows any tool to take action based on the relationships between artifacts.

3.1.5 Summary

A product line development effort produces a larger set of more complex, longer lived artifacts than does a conventional effort. The artifacts' interrelationships are multidimensional and may change along any one of these dimensions. The implications for support tools are

that they handle larger, more complex sets of diversely related artifacts. The tools must analyze the contents of the artifacts, traverse relationships among artifacts, and manage multi-directional change along these relationships.

The present generation of development tools does not meet these needs. Tool vendors should understand two important factors unique to the product line environment:

1. Tools adopted by a product line development effort will be used longer than in a conventional environment.
2. Product line projects are more reliant on tools and will demand better service. The complexity of the artifacts and the relationships among them make it essential that there be automated support for managing these artifacts.

3.2 How Can Tools Be Used to Support the Activities in Product Line Efforts?

A second working group focussed on how tools can be used to support the activities involved in software product line efforts. The discussions explored two general categories in which tool support is needed: (1) managing the commonality and variation among the assets and products in the product line and (2) managing the evolution of assets and products.

3.2.1 Managing Commonality and Variation

The group identified four items for discussion in the area of managing commonality and variation. These were feature-based builds, identifying potential core assets, dependency tracking, analysis/design/requirements/traceability, and test case management.

3.2.1.1 Feature-Based Builds

Feature-based building is a technique that allows the product system builder to specify just the features that are desired in a product and then automatically retrieve the correct combination of core assets and parameters to build the product.

The tools that would be used to manage a feature-based build are a combination of a database tool and a configuration management tool. The specifics depend on how features are actually implemented. One approach is to implement features via components, using different components in the different products to achieve variations on the same functionality, or to use parameters, inheritance, or delegation on components to achieve the variations in functionality. In this case, the mapping between the features and the components and component parameters, or hierarchical path, is stored in the database.

Another approach to feature management is to use architecture parameters (as embodied, for example, in a domain-specific language). In this case, a compiler for this language becomes an additional tool and the entries in the database become language snippets in the domain-specific language or parameters to specifications in the domain-specific language.

To build for a product system, one would query the database using both a list of the features desired and the versions of these features. The response to the database query would then be a list of components with associated parameters or subclasses that is passed to the configuration management system to generate the build. In the case of having architectural parameters, the output is passed both to the domain-specific compiler and to the configuration management system; the two of them must then coordinate to generate not only the correct features, but also the correct versions of the features.

3.2.1.2 Identifying Potential Core Assets

There are two basic techniques that were discussed to identify potential core assets. Both involve "post facto" types of analysis. That is, various products are produced and then they are analyzed to determine what within them should be considered a core asset.

The first technique involves tagging code as it is generated. The tags can be identified by using a scan tool that examines the code within the various products and determines what code was used. Such a tool needs to be coupled to a data analysis facility so that the tags can be categorized and recorded based on the products.

The second technique involves reconstructing the actual products using reengineering and semantic analysis tools. Again, the technique is to generate the actual products, analyze them with reengineering tools (typically source reengineering tools), and compare the results among the various products to determine the commonality.

3.2.1.3 Dependency Tracking

It is important when managing variability and commonality to determine the dependencies among the components. A particular feature may be implemented in one or more components, and these components may depend on additional components. Understanding and managing this dependency tree is important to reduce the impact of changes.

There are both implicit and explicit techniques for managing the dependency trees of various products. The implicit technique is to build the products and then build the trees from the components that were contained within the products, using the information generated by the configuration management and build tool.

The explicit technique is to record this information prior to build. It can be recorded in the same database used to manage feature builds or it can be recorded separately through use of UML™, XML, or interface description language (IDL). The use of XML or IDL requires components to declare their interfaces explicitly and then all components can be examined to identify the components on which they depend. The use of UML™ requires an explicit modeling step within the development process. The technique best suited to a particular organization depends on the organization's development process to determine how dependency information can be accumulated with the least amount of disruption.

3.2.1.4 Analysis/Design/Requirements Traceability

There was general consensus that traceability between requirements and design should be managed by the individual tools used for these functions. No special traceability tool makes sense. What this means, of course, is that there must be some compatibility between the tools for various functions, but this is a universal problem.

Within this context, tracing variation represents special problems. A requirement for a product, for example, may be derived from a requirement for a product line, or it may be a specific requirement for that product. Requirements tools need to be sensitive to this distinction, as do tools for analysis and design.

3.2.1.5 Test Case Management

Testing should proceed separately for core assets and for individual products. There should be a full suite of test cases suitable for core assets and individual suites of test cases for each product. Test management tools have tags that can be associated with test cases that enable the management of these different test suites.

3.2.2 Managing Evolution

Both core assets and individual products are subject to change with time. The general concern is to prevent the occurrence of "architectural drift." Architectural drift describes the situation when the architecture for the products differs from the product line architecture or the product line architecture changes over time without a disciplined process that manages the evolution and makes it explicit. Architectural drift is a phenomenon that occurs with single system evolution as well as with product lines, but the consequences are more severe with product lines due to the number of system products that are evolving. Evolution will occur, but the evolution must be explicit and managed. We discussed three topics with respect to managing evolution: enforcement of policy decisions, after-the-fact examination to verify conformance, and release management.

3.2.2.1 Enforcement of Policy Decisions

A level of authorization can be introduced within the configuration management process and supported by the configuration management tool. The individual software engineer is allowed to set parameters within a build, but changing components can be restricted during builds.

This process adjustment occurs during single system development, as well. The difference with product lines is the organizational structure that defines the policy and the roles of the individuals who are allowed to make changes. Discussion of organizational alternatives was outside the scope of this tools breakout group but is addressed in the SEI Product Line Practice Framework in the following two practice areas: (1) Operations and (2) Achieving the Right Organizational Structure.

Another technique is to build allowable variations into a domain-specific language. Then anything that can be expressed within that language is allowable, but making changes beyond those allowed in the language would require intervention by other groups.

Other techniques are possible. One is to ensure that no modifications are allowed to derived objects and that the first class objects are sufficiently powerful to support all possible variations. Another is to have a constraint checker that verifies that modifications are within the scope of particular rules that have been established.

3.2.2.2 After-the-Fact Examination to Verify Conformance

Determining violations of policy after the fact can be done with a tool for verifying architectural conformance. Such tools are still very experimental. It could also be done with tools for checking coding standards. Such tools exist but are limited in what they can detect.

3.2.2.3 Release Management

One of the circumstances that can arise with product lines is the nearly simultaneous production of two products. Due to tight schedules governing the release of these products, some changes may be introduced into the core assets for incorporation into one product and not into another. The configuration management system must be sufficiently sophisticated to allow a distinction between the versions of the core assets that are used by different products.

Problem reports and defect tracking also must be done differently for product lines than for single systems. When defects arise within a single product, the extent of the defect within the product line must be determined. This argues for a sophisticated defect-tracking system that understands the concept of product lines and releases with differing ranges of impact. Spreadsheets, dependency charts, and various decision support tools can be used by the configuration control board to support release management.

3.2.3 Risks

Finally, the group discussed the following two risks associated with the use of tools in a product line effort.

1. Using an inappropriate synchronization model for the particular business model of the organization is a risk to the product line effort. The synchronization model depends on the ownership of the core assets. Synchronizing releases of the products with releases of the core assets depends on close cooperation among the various owners. In a business model where the releases of the core assets are not coordinated with the schedule of the product development, it is possible to have a great deal of disruption in the development of products. Use of a visualization tool will help ameliorate the risk.
2. Many of the tools normally used often allow extensions so that they can be customized as desired. This type of facility is used to accommodate special product line needs with existing tools. The risk is that the extension of the tools becomes time consuming and a heavy maintenance drain. New releases of a tool may invalidate the particular extensions

used by an organization and cause both rework and degradation in the tool support needed to sustain the product line operation.

3.2.4 Summary

Tools can be used to help manage two of the most challenging aspects of software product lines: the commonality and variation among the assets and products in the product line, and the evolution of those assets and products. Specifically, tools can assist with feature-based builds, identification of potential core assets, dependency tracking, analysis/design/-requirements/traceability, and test case management. All of these activities are essential to managing commonality and variation. In the area of evolution, tools can help enforce technical policy decisions, verify conformance, and directly enable release management.

3.3 What Variety of Tools Is Needed for Product Lines?

The third working group discussed the variety of tools needed to support a product line. Their goal was to examine the areas of a product line development effort for which tool support is needed, recognizing that such an effort entails both asset development and product development. This group tried to determine how tool support for product lines should be similar to or different from that used in conventional development efforts. In particular, the group attempted to outline tool capabilities and risks that are particular to product lines.

The group started by thinking about the practices that constitute a product line approach and how tools might support these practices. The SEI Product Line Practice Framework identifies a set of product line practices that may warrant appropriate tool support. Many of these practices correspond to practices that are also present in conventional development, many of which have tool support. However, there are other practices that are not present in conventional development. This distinction was used to structure the discussion, distinguishing “first-order” tools, which support practices that have a role in conventional development, from “second-order” tools, which support practices that occur only in product line efforts.

3.3.1 First-Order Tools Analysis

The group examined a broad range of product line practice areas for which automated support exists in conventional development. The intent was not to be exhaustive, but to gain consensus on the current state of existing tools from the perspective of product line needs. Tools in this category provide support for “traditional” software development efforts in most areas of software engineering:

- requirements analysis
- architectural design
- component specification
- component internal design and coding

- product building (component retrieval, configuration, compile, and link)
- verification (including testing and reviews/inspection)
- configuration management/change control
- project management
- traceability
- documentation (user/customer or to supplement tool-generated documentation)
- economic modeling
- application-specific commercial off-the-shelf infrastructure (e.g., database, middleware, protocol stacks)
- team communication and collaboration
- prototyping/rapid application development, modeling, and simulation
- measurement (collection and analysis)
- reengineering and reverse engineering (e.g., artifact analysis)

These "first-order" practice areas are supported, some well and some poorly, by a variety of tools. The workshop participants had used some of these tools in a product line effort. Significantly, no one could identify any tools used in conventional development efforts that did not have a similar role in some aspect of a product line effort. That is, if an organization has selected a tool to support a conventional development activity, such as requirements analysis, then that same tool is likely to be appropriate for use in at least one analogous product line asset development or product development activity.

The group concluded that first-order tools remain relevant (and necessary) for product lines; there is nothing about product lines that obviates the need for tool support in these areas. The second conclusion was that virtually none of the tools addresses the need to represent *multiple* products simultaneously. First-order tools by their very nature tend to support a single product. Allowing multiple products is either not supported at all, or requires one to encode the multiplicity in terms of single product variants.

3.3.2 Second-Order Tools Analysis

Having identified by consensus a representative set of first-order tools, the group next examined aspects of product lines that require special support beyond what is handled by today's tools. The principal desired capability for second-order tools is supporting multiple products simultaneously. Moreover, beyond simply *permitting* multiple products, one would also desire capabilities to *relate* them in some unified fashion. For example, one typically would like to understand in what ways the products differ and in what ways they are alike.

Aspects of product lines requiring support beyond the capabilities of today's tools include

- Environment building: Create an infrastructure to support the development of the product line. The tools in this category include

  - tool-building tools (user interface or language construction, simulation, testing)
  - tool-integration tools (scripting, customizing tool features, bridging/glue code, wrappers)
- Process modeling: Establish the structure and interrelationships of the various process elements of core asset development/acquisition and product development/acquisition. The iteration inherent in product line activities must be supported by any realization of the process model.
- Scoping a product line and capturing assumptions made during the product line development: The scope of a product line determines which products are members of the product line. The product line development effort can be viewed as successive refinement of the initial business case for the product line. The core asset base will evolve through iterative development of intermediate work products that embody decisions and assumptions about the common capabilities of product line members, and their allowed variability. Automated support should facilitate the capture and representation of commonality and variability in a manner that enables the scope of the product line to be determined and refined.
- Product derivation: Creating products from product line assets means that tools that support the production strategy are required. For example, products may be generated from core assets or *assembled* from core assets; the tools used should support the strategy chosen.

In some cases, relevant tools are currently available for use in creating software development tools (e.g., tool-builders) but are not generally used in developing application products. A notable omission from this list is domain analysis. The group consensus was that there are currently no tools specifically for domain analysis. Instead, tools created for other similar purposes, such as requirements analysis, are adapted to the needs of product line domain analysis. The primary adaptation required, as noted regarding first-order tools, is to encompass a set of similar products rather than one product only.

3.3.3 Risks

The main complication in providing tool support for product lines comes from the distinction between support for building a single product and support for building many similar products. Current tools geared toward single-system development efforts lack the capabilities to support the development of core assets tailorable to different uses or the multiple products of a product line.

There are also risks associated with these tools, some pertaining to the capabilities of tools in general (although often made more severe by the nature of product line development), and some specific to their use for product lines.

The general risks are described below:

- The steep learning curve associated with most tools. For example, it may take a relatively short time to learn the essentials of a sophisticated configuration management tool, but it may take much longer to master it so that it can be productively used to handle the more extensive needs of product line development.

- Lack of interoperability between different tools. Each tool has its own model of operation, which makes it difficult to integrate outputs from different tools and difficult to handle incompatible data formats (e.g., requirements from several different sources, including legacy systems). While this is not specific to product lines, the lack of existing *integrated* tool suites makes this problem more severe than for single product development.
- Insufficient or inappropriate process assumptions built into a tool. Although tool builders may try to be neutral about process, often they make assumptions about a tool's use, thereby affecting the way the tool operates. This may not be compatible with any particular organization's development process and may force the organization to accommodate an inferior process just to make the tool usable. For example, the organization may wish to follow an iterative, incremental development process that creates intermediate, partially completed work products in each iteration. The tool may assume a waterfall process and require "completion" of a phase before proceeding to the next phase. (In a product line context, one manifestation of this problem might be an inability to analyze a product line architecture that, during its creation, may be inherently incomplete or ambiguous.)
- Mismatch between a selected tool and the problem to be solved. A tool may simply be the wrong tool for a job because of a conceptual mismatch between it and the problem domain. Alternatively, the tool may provide only an incomplete solution to the problem. For example, several of the participants have used object-modeling tools to represent product line architectures. However, object models (and their associated tools) address only some aspects of architectural representation.
- Lack of control over evolution of a tool. A tool (or any shared resource) may evolve in a direction that makes it less and less suitable for the task to which it was initially applied. Alternatively, the nature of a problem may evolve in a different direction. This may result in a lack of particular capabilities needed to support the evolving requirements of a development effort. For example, a tool may support both analysis and design, but, because of a business decision on the part of the vendor, evolve to support design at the expense of analysis.

Some risks specific to product lines are described below:

- Longer development cycle due to a lack of true support at the product line level. For example, the inability of tools to represent multiple products' information in a connected/unified form (specifically with respect to commonalities and variabilities) may undermine the whole concept of creating multiple products from the same core asset base.
- Mismanagement of product line requirements. The requirements tool used may not accommodate
  - the range of variation in specific requirements (that bound the set of products included in the product line family)
  - constraints on the possible combinations of requirement variations
  - verifying what requirements are excluded (e.g., "no member of this product line will ever be capable of X")
- Loss of flexibility in the product line. Controlled variability is essential for creating a product line that is flexible enough to accommodate changing customer needs. It is important to be able to capture and reason about permitted variability early in the life cycle. Poor automated support in this area may, for example, lead to

  - the inability to express variability in architecture, particularly in graphical form
  - the inability to trace requirements variability to architecture variability
- Defective core assets. For example, inadequate options for the analysis of architectural properties such as reliability, performance, security, etc., may lead to a flawed product line architecture, with the downstream consequences of costly, time-consuming repairs to design flaws.
- An out-of-control production and release schedule. For example, insufficient configuration management support—especially for capturing and correlating data about changes across multiple products—has a major effect on an organization's ability to field products quickly and respond to requests for changes.
- Time wasted on corrective maintenance. For example, a tool may provide insufficient verification support—especially for leveraging testing at the product line level and for relating discovered errors in a product to its source. To fix an error at its source means to determine if an error in a particular code component was introduced into that component alone by a product developer, was caused by incorrect instantiation of a product line asset, or was already present in the product line asset from which the code component was instantiated.

3.3.4 Summary

Ultimately, the goal of using tools in a development effort is to automate the efficient creation of software products. The product line context means that the tools employed need to support an iterative, incremental process of creating or acquiring core assets and using those assets to build products according to a production strategy. Tools that support practices within conventional software development efforts (e.g., requirements analysis) have their counterparts in product line development, but the multi-system context and long life of a product line means that additional capabilities are needed. Because of the lack of such capabilities in today's tools, there are some risks associated with tool support for product lines over and above the usual risks of automating software-development efforts.

4 Summary

The SEI's Fourth Product Line Practice Workshop explored the area of tool support for product lines with representatives from technically sophisticated organizations having direct experience in software product lines. The presentations and discussions underscored both the dire necessity of tool support to succeed with software product lines and the inadequacy of available tools.

There are differences in a product line environment that not only require automated support, but require robust and specific kinds of support. The core assets used to build products have to express the commonality and variation among products. Traceability support, evolution support, and interoperability among tools are tantamount to success. Because adequate tool support is not currently available, organizations must modify existing tools and in many cases make their own. In many situations, they simply do without. As a result, organizations take on risks that can undermine their product line efforts and at the very least consume resources to work mitigation strategies.

The participants at this workshop were impressed with the creativity each had used in overcoming obstacles stemming from the lack of truly supportive tools, but were frustrated with the lack of tools that forced them to expend their creativity in this way. What will motivate tool vendors to create and market the tools that are so needed for product lines? The following were offered as suggestions:

- Provide firm data that point to future sales for particular tools or a whole tool set that will support product lines.
  - Take a survey.
  - Conduct market research.
- Encourage an open source approach for those who have developed in-house tools.
- Educate tool vendors that there are needs and problems today addressing those needs.
- Promote standards for product line tools.
- Point out lost cost add-ons to existing tools that would be quick to get to market.
- Find champions in vendor organizations.
- Organize panels on this topic at upcoming conferences.
- Create an executive summary of this report and send to tool vendors.
- Work to penetrate and influence tool user groups.

The participants agreed to work together to precipitate some change in the area of tool support. The SEI was encouraged to continue to grow both the information base and the community interested in software product lines, and in particular, tool support for product lines.

The results of this workshop will be incorporated into the framework,[5] which will continue to be refined and revised as the technology matures and as we continue to receive feedback and to work with the growing community of software engineers championing a product line approach. If you have any comments on this report, if you would like to help in the quest to motivate better tool support for product lines, or if you are using a product line approach in the development of software-intensive systems and would like to participate in a future workshop, please send electronic mail to lmn@sei.cmu.edu.

[5] Version 2 of the SEI's Product Line Practice Framework is on our Web site <URL: http://www.sei.cmu.edu/plp/framework.html>.

References

[Bass 97] Bass, Len; Clements, Paul; Cohen, Sholom; Northrop, Linda; & Withey, James. *Product Line Practice Workshop Report* (CMU/SEI-97-TR-003, ADA 327610). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997. Available WWW <URL: http://www-preview.sei.cmu.edu/publications/documents/97.reports/97tr003/97tr003abstract.html>.

[Bass 98a] Bass, Len; Clements, Paul; & Kazman, Rick. *Software Architecture in Practice*. Reading, Massachusetts: Addison-Wesley Longman, Inc., 1998.

[Bass 98b] Bass, Len; Clements, Paul; Cohen, Sholom; Northrop, Linda; Smith, Dennis; & Withey, James. *Second Product Line Practice Workshop Report* (CMU/SEI-98-TR-015, ADA 343688). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998. Available WWW <URL: http://www-preview.sei.cmu.edu/publications/documents/98.reports/98tr015/98tr015abstract.html>.

[Bass 99] Bass, Len; Campbell, Grady; Clements, Paul; Northrop, Linda; & Smith, Dennis. *Third Product Line Practice Workshop Report* (CMU/SEI-99-TR-003, ADA 361391). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. Available WWW <URL: http://www.sei.cmu.edu/publications/documents/99.reports/99tr003/99tr003abstract.html>.

[Clements 99] Clements, Paul; Northrop, Linda M.; et al. *A Framework for Software Product Line Practice, Version 2*. Available on the WWW <URL: http://www.sei.cmu.edu/plp/framework.html>, August 1999.

[Weiss 99] Weiss, David M. & Lai, Chi Tau Robert. *Software Product Line Engineering.* Reading, Massachusetts: Addison-Wesley, 1999.

[Withey 96] Withey, James. *Investment Analysis of Software Assets for Product Lines* (CMU/SEI-96-TR-010, ADA 315653). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996. Available WWW <URL: http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.010.html>.

Glossary

application engineering	An engineering process for developing software products from partial solutions or knowledge embodied in software assets
domain	An area of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that area
domain analysis	Process for capturing and representing information about applications in a domain, specifically common characteristics and reasons for variability
platform	Core software asset base that is reused across systems in the product line
product family	A group of systems built from a common set of assets
product line	A group of products sharing a common, managed set of features that satisfy specific needs of a selected market or mission area
product line approach	A system of software production that uses software assets to modify, assemble, instantiate, or generate a line of software products
product line architecture	Description of the structural properties for building a group of related systems (i.e., product line); typically the components and their interrelationships. The guidelines about the use of components must capture the means for handling variability discovered in the domain analysis or known to experts. (Also called a reference architecture)
product line system	A member of a product line
production system	A system of people, functions, and assets organized to produce, distribute, and improve a family of products. Two functions included in the system are domain engineering and application engineering.
software architecture	Structure or structures of the system, which consists of software components, the externally visible properties of those components, and the relationships among them [Bass 98a]
system architecture	Software architecture plus execution and development environments

software asset A description of a partial solution (such as a component or design document) or knowledge (such as a requirements database or test procedures) that engineers use to build or modify software products [Withey 96]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (LEAVE BLANK)	**2.** REPORT DATE **February 2000**	**3.** REPORT TYPE AND DATES COVERED Final
3. TITLE AND SUBTITLE **Fourth Product Line Practice Workshop Report**		**5.** FUNDING NUMBERS C — F19628-95-C-0003
6. AUTHOR(S) **Len Bass, Paul Clements, Patrick Donohoe, John McGregor, Linda Northrop**		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		**8.** PERFORMING ORGANIZATION REPORT NUMBER **CMU/SEI-2000-TR-002**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		**10.** SPONSORING/MONITORING AGENCY REPORT NUMBER **ESC-TR-2000-002**
11. SUPPLEMENTARY NOTES		
12.A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		**12.B** DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

The Fourth Software Engineering Institute (SEI) Product Line Practice Workshop was a hands-on meeting held in December 1999 to share industry practices in the area of tool support for software product lines, to explore the technical and non-technical issues involved, and to evolve the SEI Product Line Practice Framework. This report synthesizes the workshop presentations and discussions, which described practices and issues associated with tool support for software product lines.

14. SUBJECT TERMS **product line practice, Product Line Practice Framework, software assets, software engineering, software product lines, tool support**	**15.** NUMBER OF PAGES **36**
16. PRICE CODE	

17. SECURITY CLASSIFICATION OF REPORT	**18.** SECURITY CLASSIFICATION OF THIS PAGE	**19.** SECURITY CLASSIFICATION OF ABSTRACT	**20.** LIMITATION OF ABSTRACT
UNCLASSIFIED	**UNCLASSIFIED**	**UNCLASSIFIED**	**UL**

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102