Software Product Families in Europe: The Esaps & Café Projects

Frank van der Linden, *Philips Medical Systems*

European companies have been cooperating for seven years on product family development. The size and scale of the projects have increased dramatically, and funding now comes through the ITEA framework from local governments. Here is an overview of these projects, their results, and how they differ from other product line efforts.

The main drive for Europe's industrial-cooperation projects on product family development is business need. The participating companies have realized that only through significant reuse could they increase software productivity, and for reuse to happen, it had to be planned and organized.

In 1995, the small Esprit project ARES (Architectural Reasoning for Embedded Systems)[1] began with a focus on architecture support for developing product families. Near the end of ARES, another Esprit consortium started investigating family development processes in the Praise project. When ARES finished, both ARES and Praise forces began to cooperate on a larger scale within the Information Technology for European Advancement framework (www. itea-office.org). Established in mid-1999, ITEA is an industry-driven, multinational strategic research and development program designed to promote the development of embedded and distributed software and related software engineering technologies. Because ITEA projects last for at most two years, there has been a sequence of such efforts. The first, Esaps (Engineering Software Architectures, Processes, and Platforms for System Families), recently finished, and the second, Café (Concepts to Application in System-Family Engineering), has just begun. The research in Café will extend the Esaps results by providing methods and processes that support independent life cycles of products and of systems using these products. In short, Café will tie the separate concepts of Esaps into a unified whole covering a product family's entire life cycle. A follow-up project is already being planned.

Companies in the ITEA projects (see Figure 1) are working on a variety of embedded systems, including medical imaging, mobile phones, flight control, utility control, supervision and management, financial services, and car electronics. Because they are still introducing product family practices in their organizations, not all conclusions are definite. The five major partners—Philips, Nokia, Siemens, Thales, and Telvent—introduced product family development in software organizations of more than 100 people. Several other companies and research institutions are involved in parts of the program: they introduce family development on smaller scales, focus on special topics, or act as consultants for other companies.

0740-7459/02/$17.00 © 2002 IEEE

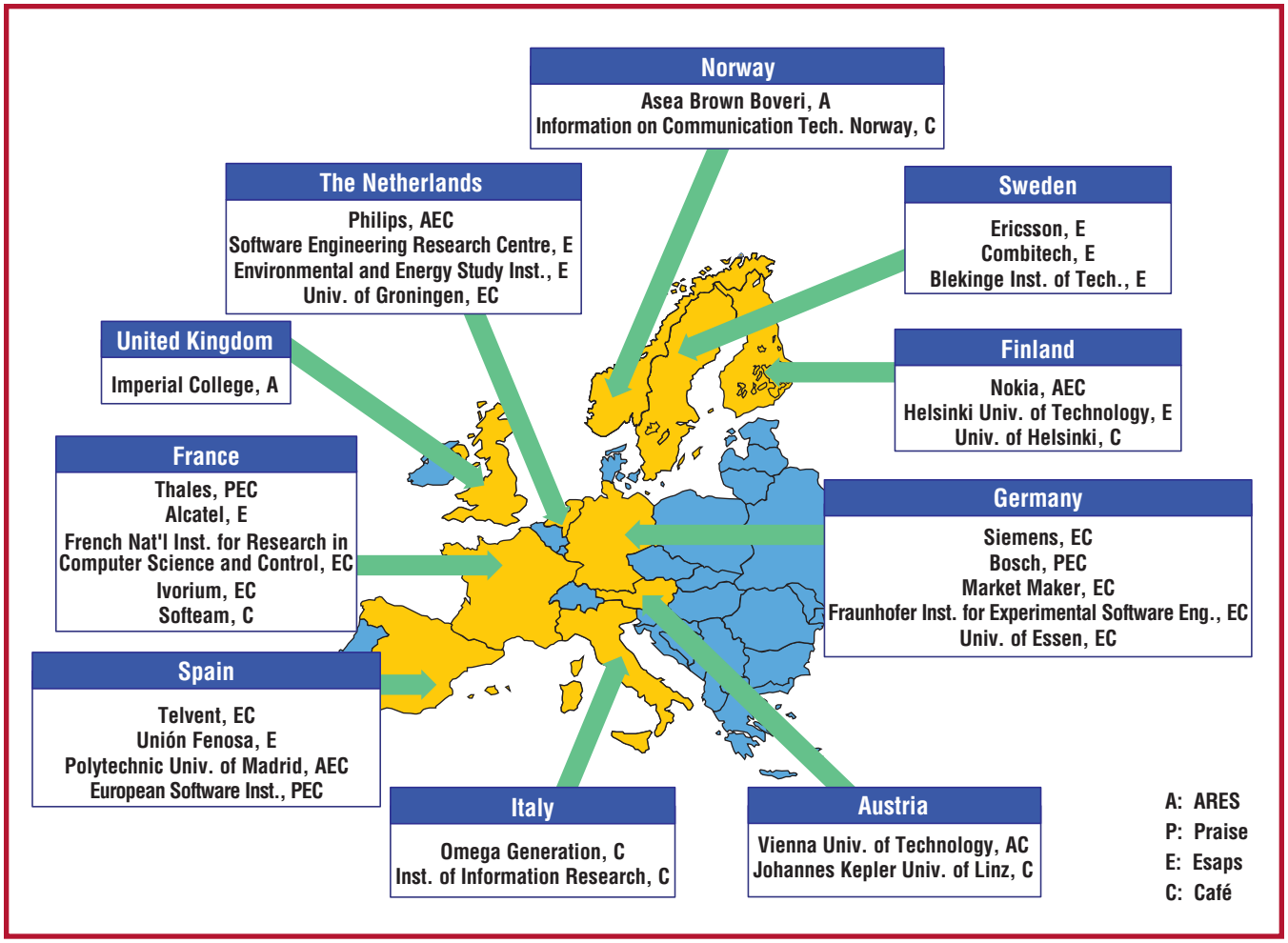

Figure 1. The companies and countries involved in ITEA projects. Each partner's name is followed by a code denoting its involvement in the projects.

Product line, product family, product population

You might notice that we use the term *product family* or *system family* where others use *product line*. This is because the US and European communities in this field worked independently until a first meeting in Las Navas, Spain, in November 1996. By that time, two sets of terminology were already established.

Moreover, certain European companies use *product line* to indicate something different—a set of related, commercial products that appear similar to users but often are built with different technologies. For example, product lines in consumer electronics include televisions, VCRs, DVD players, audio receivers, CD players, audio amplifiers, and so on. These products have similar housings and user interfaces (for instance, buttons, dials, and LEDs), but their internal technical details might differ a lot. We use *product family* to describe a collection of products that are based on the same technology—for instance, a collection of TVs based on the same software architecture. Often products in the same product line are in different product families, and vice versa. Because good communication between the architects and the marketing department is important in product family development, using *product family* or *system family* is easier.

Rob van Ommering introduced the technical term *product population*.[2] This term denotes a collection of related systems based on a similar technology but with many differences. For instance, the complete set of consumer electronics products mentioned earlier can form a product population.

Family development concerns

Setting up product family development means taking several concerns into account. The acronym BAPO, introduced by Henk Obbink of Phillips Research, covers the main issues. BAPO stands for

- *Business:* the way the resulting products make a profit
- *Architecture:* the technology needed to build the system
- *Process:* responsibilities and dependencies during software development
- *Organization:* the organization in which the software is developed

Decisions made for one of these concerns affect them all. Thus, such decisions must be made carefully.

System family development started with the implicit idea that a good family architecture might benefit business. In other words, by dealing with diversity in a managed way, we might serve a larger, more predictable market segment. If we are going to initiate an architecture design and implementation, we need a process to determine all the actions needed. We also need an organization to implement the whole. Then we can address the business consequences in light of the results obtained.

The two main technical goals of the sequence of projects are to improve in both development paradigms and reuse level. In the case of our projects, the main architectural improvements that will influence the designs are developments in the computing platforms, distribution and communication, the development environments, and the software development paradigms. In practice, this means that we move toward component-based development—that is, varying products based on a single component-based platform (see Figure 2). The platform is the basis of the product family. Developers design, build, and test from a selection of the platform's assets. If they cannot obtain the assets they need from the platform itself, they must develop them. Later, the developer might integrate these new, single-product assets into the platform. The idea is simple but the practice is hard, and this relates to all BAPO concerns. The graph in Figure 3 shows the reuse-level terms introduced by Ivar Jacobson, Martin Griss, and Patrik Jonsson.[3] To move from bottom to top along the *y* axis means that an increasing number

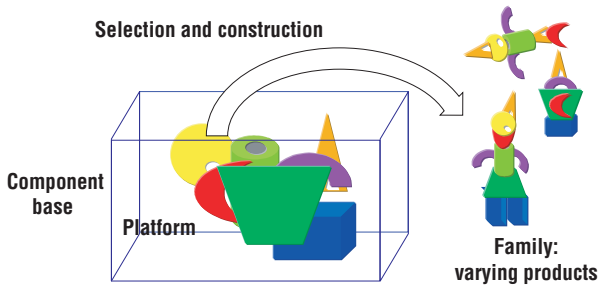

Figure 2. Component-based platform as the basis for a family. Reusable assets might include interfaces, designs, patterns, requirements, supporting infrastructure, and test cases.

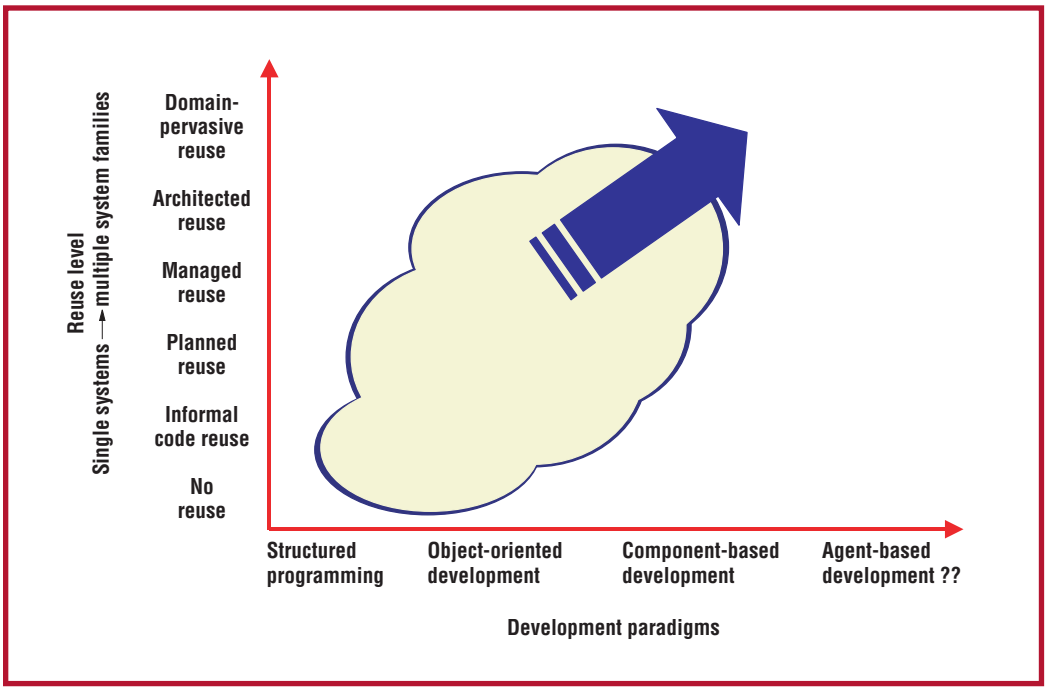

Figure 3. Moving from single systems to multiple system families.

Table 1

Project topics

	Business	Architecture	Process	Organization
ARES		Dealing with variation Architecture description Resource management qualities	Recovery from legacy systems	
Praise		Domain-specific architectures Family architecture Development tools	Family development practices Variability and commonality Traceability between assets Architectural decisions	
Esaps	Scoping	Domain analysis Aspect analysis Family requirements Architecture glossary Commonality and variability Reference architecture Platform and components	Architecture assessment Architecture recovery Domain analysis Aspect analysis Family development process frameworks Requirements modeling and traceability Change management Evolution support Variant configuration and derivation	Platform and component development
Café	Business and market analysis Scoping Family development transition and adoption	Requirements engineering Heterogeneous platforms COTS use Design for quality Development tool support Test modeling Validation	Asset management Traceability Change management and impact analysis Family transition and adoption Configuration and version management Product derivation Family evolution Test strategy and methodology Validation	Asset management Validation and testing Product-line transition and adoption Change management Configuration and version management Product derivation

of assets other than software—for example, designs, patterns, requirements, test specifications, and test results—become reusable. In addition, planning and design of reuse become necessary. The product family approach deals with architected reuse.

Our sequence of projects addressed various software development issues. The ARES project considered mainly architecture concerns for system families, and the Praise project considered mainly process issues. Praise introduced a reference process model that we use in Esaps and Café. The Esaps project concentrated on creating and managing family assets but also did some work on the architecture and the organization. Café also has a strong process focus, but it spends more effort than the previous projects on organization and business issues. Café takes into account issues at the very early and later stages of development related to requirements engineering and testing.

The companies working in the different projects agreed to use a particular set of best practices. Table 1 groups these practices according to the development issues discussed later.

Business

We began the sequence of projects with an intuition that family development was good for business, but we didn't address the first business concern, scoping, until Esaps. Now, business concerns are an important topic in Café. The scope determines the product family's range, and scoping decisions have important business impact. If the scope is too wide, development becomes too costly; if it is too narrow, the product family cannot serve the market demand. We must choose project objectives and a domain focus that are appropriately scoped and aligned with the broader strategic needs of the market and the stakeholder organization. Scoping is not an initial activity, but it must be performed over the course of the family's lifetime, because new market demands might give rise to new scoping solutions.

Esaps participants distinguished three kinds of scoping:

- *Product family scoping*: define the product portfolio
- *Domain scoping*: identify the boundaries to relevant domains

- *Asset scoping*: identify reusable elements

Esaps activities dealt only with the latter two. Domain scoping is related to domain modeling activities; asset scoping is related to feature analysis, which determines a product's commercial features. We used an initial tool prototype to help us make scoping decisions.

The Café project must answer these important questions:

- From a market point of view, when is defining a product line worthwhile?
- When and how should we introduce a product family approach?
- How do we integrate existing processes with the new ones derived from the product family paradigm?

To evaluate these questions, we need an economical analysis model. To make scoping more effective, we want to relate it to the business model. Finally, Café will deal with the problems of introducing a family development process into an organization. Emphasis will be on which organizations are suitable for product family development, and how traditional organizations can move toward product family processes. The project will cover both lightweight and heavyweight transitions.

Architecture

The ARES project focused on architecture, as Figure 4 shows. One of the most important results was a clear definition of these architecture issues:

- *Significant architecture requirements*: The architecture should make explicit which requirements are significant for the product family's architecture. Moreover, it should be explicit whether a requirement is functional or nonfunctional (related to quality). The architecture should relate all these requirements to the design decisions made.
- *Concepts*: The architecture determines concepts, which clarifies the system's organization.
- *Structure*: The architecture determines the internal organization of the products.
- *Texture*: The architecture determines the standard solutions for implementation problems. Developers can address quality issues by using such standards.

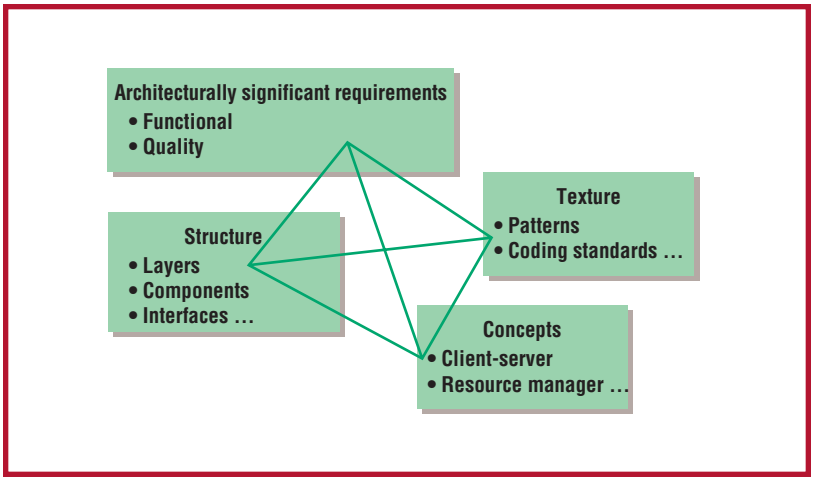

Figure 4. The ARES model of software architecture.

ARES spent much effort on the architecture description. Having the right description mechanisms eases the modeling of variability. This modeling continued in Esaps. As a result, we recognize that variability should be modeled through variation points,[3] which denote places at which the family's assets might vary. Esaps distinguished between variability in time and in space. This distinction strongly influences how the variability will eventually be implemented.

Esaps investigated the subject of variability. Designing system families requires finding a way of architecting both commonality and variability to exploit them during the tailoring process. The product family architecture, or reference architecture, defines the components (mandatory, optional, and alternative), component interrelationships, constraints, and guidelines for use and evolution in building systems in the product family. Consequently, the reference architecture must support common capabilities identified in the specification, the commonality, and the potential variability within the product family. We then use the reference architecture to create an instance of a particular architecture for a new variant.

Product line architecture modeling in Esaps resulted in separating the concepts *variability in the large* and *variability in the long term*. This is a useful separation, because it relates to the type of equipment and to the market. Moreover, the variation patterns used differ for these categories.

The Esaps project recognized the most important reusable assets:

In a product family context, each component must carry much more information than traditional software packages.

- *Requirements, with variation points*: stating stable and variable (planned) product properties for all family members
- *Domain model*: describing all the entities that are important for all systems in the family
- *Architectures*: determining how to configure different software assets to build systems that satisfy the quality requirements
- *Patterns*: used in solving quality and variability issues during product development
- *Design decision model*: used in determining how to derive a product based on specific requirements
- *Software components*: implementing the functionality
- *Interfaces between components*: providing more stability than using the components directly; often enabling different implementations of the same functionality
- *Test cases, with variation points*: used in testing products in the family
- *Product documentation*: used by the system's end users

To build systems right, we must develop our platforms and components carefully. In many cases, we build the platform itself from components. In the Esaps project, we addressed component management, including identifying and retrieving assets and designing, implementing, and delivering components. We also investigated component configuration support; we found that in a product family context, each component must carry much more information than traditional software packages. The Café project continued this work, emphasizing the independence of the underlying commercially bought platform and the use of commercial off-the-shelf (COTS) software in the families.

In ARES, Esaps, and Café, attention to designing for quality was an important issue. In a product family context, designing for quality has specific challenges, because the developers must make quality predictions for all the products in the family. In ARES, we addressed traditional qualities, mainly those dealing with resource issues.

In Esaps, we gave specific attention to *aspect analysis* to deal with special system views for various quality concerns. Aspect analysis is inspired by the Building Blocks practices[4] in Philips Kommunikations Industrie in the 1980s and early 1990s. PKI had to survive in a niche market and therefore needed to be very flexible. This architecture was the basis for one of the first product families in Philips. The developers considered three design dimensions independently—structure, aspects, and behavior—and then assigned each piece of functionality a place in each dimension. The structure dimension determines the component and layer where the functionality resides. The behavior dimension determines the threads in which the functionality is executed. The aspect dimension does a high-level functional decomposition of the system. Each piece of functionality was assigned to a single aspect—for example, operational behavior, logging, initialization and termination, test management, process management, and resource management. Each component, called Building Block by PKI, implemented all aspects. Sometimes with the support of automatic code generation, we created a global design for each aspect. Esaps developers took up this idea to see whether it would apply to more general situations. Some investigated how this relates to aspect-oriented programming. In Café, attention is on using architectural styles and patterns to address quality issues at the architecture level.

Esaps developers did a lot of work on requirements modeling for product families. Within product families, requirements come in several sorts. Some requirements hold for the complete family, but others hold for only one or a few systems. Requirements might have variation points for different family members. To be able to deal efficiently with the requirements, it should be clear what kinds of requirements are necessary and useful in the family. Traceability is related to this. Requirements must be traced to family assets. In this way, we can determine which assets are necessary for which set of product requirements.

Praise and Esaps did a lot of work on traceability modeling. Esaps came with a distinction between pre- and post-traceability, determining whether traceability is related to assets in an earlier or a later stage than requirements modeling. In addition, the developers distinguished between horizontal and vertical traceability and whether

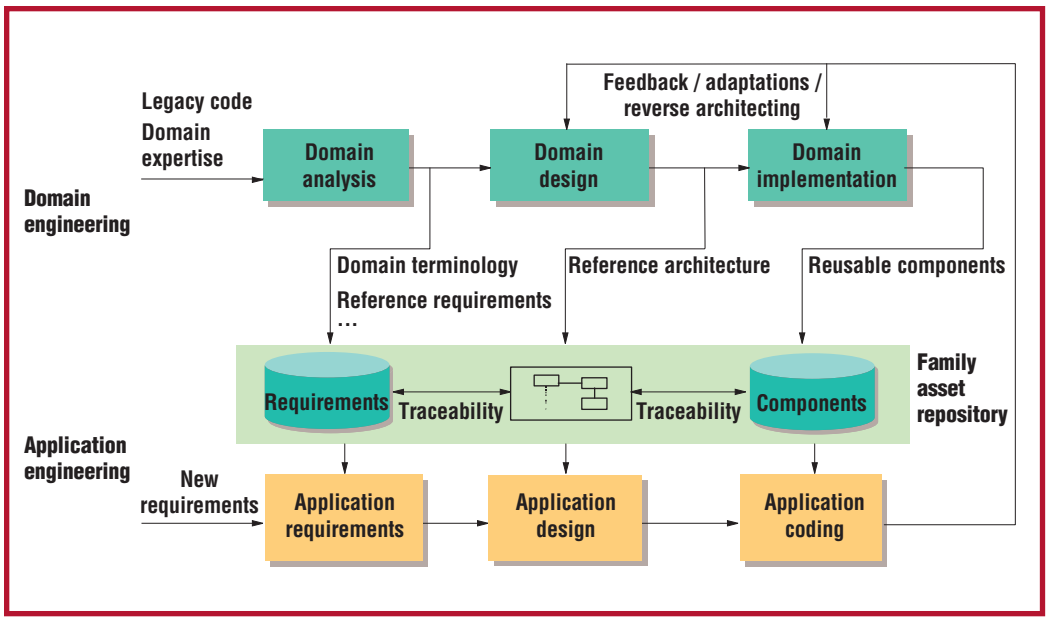

Figure 5. The Praise reference process.

traceability remains at the same abstraction level. This work continues in Café.

We studied ways (methods and tools) to support the process of determining which variants should be selected at each variation point; this is an important step in deriving the products. Esaps determined a framework for the technology and the concepts related to variability. The framework helps the developers recognize where and when variability is needed and led to patterns often related to variability. A crucial factor here is the right choice of variable-point representation.

Process

The typical software development process involves separate development for each product; instead, we wanted to introduce a development process incorporating all of a family's products. Moreover, the process had to take into account asset reuse. The advantage of this setup is that the total development cost decreases after several systems are built. The Praise project delivered an abstract reference family development process (see Figure 5) that we carried over into all activities in Esaps and Café.

There is a simple application development process for developing single systems involving

- *Application requirements*: determining what the product should be
- *Application design*: selecting components to make the product
- *Application coding*: combining components using the infrastructure and possibly additional product-specific code

The domain development process produces family assets that the product development process will reuse. This process involves

- *Domain analysis*: determining what the family is about
- *Domain design*: deciding which platform components are needed
- *Domain implementation*: building and buying components and supporting infrastructure

Two important product family issues dealt with at this point are traceability and change management, which enable developers to plan and produce systems efficiently. Traceability is connected to configuration and version management. At any moment, many system configurations are available in the field. For maintenance reasons, we need to know which versions of which assets are used in which systems. The feedback loop is essential for getting a manageable process.

The Esaps companies have implemented this process in many different forms. The development process distinguishes family asset development (domain engineering) from product development (application engineering). The asset base is the repository of family assets produced by domain engineering and used by application engineering.

We can map Jacobson, Griss, and Jonsson's process classification[3] to this model as well. Their AFE (*a*pplication *f*amily *e*ngineering) is related to the activities in the top-left hand block of Figure 5, CSE (*c*omponent *s*ystem *e*ngineering) is related to the other blocks in the top row, and ASE (*a*p-

All Esaps partners already had a collection of systems in the field that should belong to the family and that need to be maintained for several more years.

plication *s*ystem *e*ngineering) comprises the bottom row.

Esaps presented all the process frameworks used in the participating companies in the same template, so we could compare the different frameworks. We found room for improvement in all the frameworks. Esaps determined a clear set of necessary activities and work products. In all cases, domain engineering and application engineering had separate processes. Because no existing framework exhibited all the important activities, we concluded that none were sufficient for product family development.

All Esaps partners already had a collection of systems in the field that should belong to the family and that need to be maintained for several more years. Moreover, these systems contained assets that could be transformed into family assets, to be reusable in future family members. We began to define a process and tool support for incorporating the already existing (legacy) systems into the family domain, and we began to treat them as family members henceforward. Café continues on this track and has introduced the incorporation of third-party software and COTS into the family.

The Café project is based on the same Praise reference process, with a focus on the very early and late subprocesses and the major activities shown in Figure 5. The Café project's results have the form of structural rules for assets, methods, procedures, and organizational structures. We will use these results to build new applications in the families and to provide requirements for tool vendors.

Change management lets us predict the properties of variants and new family assets before building them. The Esaps project did some initial investigation of change management, and these efforts are continuing in Café. Changes in one asset of a family might affect many products in several ways, so we must be careful with changes. Guidelines and automated support are essential for identifying and supporting required changes caused by the modification of a given software artifact. These guidelines might be accomplished by specifying specific tradeoffs or constraints on specific products or product parts. Moreover, it is necessary to be able to select the requirements for a single member of the family and then to quickly select the assets needed to build that member. Esaps proposed a decision model, based on a requirement hierarchy, for deriving family members.

Depending on the specific functional or quality requirements, the architect selects the product variants to be delivered and then builds the executable system based on that selection. Esaps investigated how to manage family assets so that developers can find what they need. The project built metamodels to improve asset classification and introduced methods for effectively selecting components and interfaces in an asset base.

ARES started the first investigation of architecture assessment, the process of measuring system properties based on an architecture description. The project performed a small assessment related to timing behavior and proposed tool support for analyzing the timing behavior. A more complete assessment case study within Esaps resulted in some additions to traditional architecture assessment. However, we did not find much difference from traditional architecture assessment.

Domain analysis is one of the basic activities in product family development. All systems in the family belong to the same domain or a small set of domains, and the result of the domain analysis, the domain model, is usable for all these systems. The domain model is crucial for the family architecture, because the complete family architecture is based on the domain. Only when the developer knows the domain's boundaries can he or she efficiently determine the domain model.

ARES experimented with and had some success in recovering architecturally useful information from existing assets, informal documentation, and interviews. Esaps continued this work in the automatic verification of conformance to architectural rules during development, and the work continues in Café.

Tool support is crucial. Product family development needs tool support to manage the assets, model the domain, support traceability, and help the architects and developers easily do the right things. In addition, the (semi)automatic derivation of products from specific product requirements would be very helpful. No good tool support exists for this. In ARES' focus on techniques for product family architecture, it found that a simple language works best. However, most

commercial tools support complex languages that divert the attention of architects to issues that are important only in later stages of development. Esaps produced a collection of tool requirements. Café will get a more complete picture of requirements for tool support.

Organization

The projects have not done much work on organization yet, but Café will investigate this aspect. During Esaps, we investigated only briefly the organizational consequences of a product family development approach. We concluded that separate development groups should work on family engineering and product engineering. In fact, division into three development organizations might be a good idea.[3] However, we found that good experiences also occur when we combine all responsibilities and split the departments orthogonally to the process categories. So, we concluded that it is still not clear when and why a development organization should be separated into family and product development departments.

While the European movement toward product family engineering took place, researches in the US founded the SEI Product Line Initiative.[5] This initiative had the same objectives: improving and introducing product families (product lines) into industrial organizations. Apart from this, we know of no other large-scale product family engineering efforts. In contrast to the SEI initiative, the companies involved initiated the European projects. This was based on their own experiences and their own economic need for product families. The European movements joined forces to learn from each other. The founding companies selected research institutes according to their added value for the initiative.

The SEI initiative gives general guidelines on best practices for many areas. It puts a lot of work into the management process, which the European projects only partially address. In the European projects, we found many cultural differences among organizations, which led to different emphases on different management issues. Moreover, we are acting in a bottom-up way. We try to learn from each other and adopt each other's best practices. We do not want to force the participants to take over practices that do not fit their culture. This means that the European projects came up with a large variety of solutions to the same problem, in contrast to the SEI initiative, which promotes a single framework.

To bring the rest of the world into the discussion, we have been running a sequence of workshops on product family engineering.[6–8] During the first workshop, we came in contact with the SEI Initiative participants; now, we meet each other several times a year at conferences and workshops and exchange ideas for improvement. We feel we benefit from this contact, so we will continue to present our workshops.

References

1. M. Jazayeri, A. Ran, and F. van der Linden, *Software Architecture for Product Families,* Addison-Wesley, Reading, Mass., 2001.
2. R. van Ommering, "Beyond Product Families: Building a Product Population?" *Proc. Software Architectures for Product Families*, Lecture Notes in Computer Science 1951, Springer-Verlag, Berlin, 2000, pp. 187–198.
3. I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse,* Addison-Wesley, Reading, Mass., 1997, p. 21.
4. F.J. van der Linden and J.K. Müller, "Creating Architectures with Building Blocks," *IEEE Software*, vol. 12, no. 6, Nov. 1995, pp. 51–60.
5. P. Clements and L. Northrop, *Software Product Lines,* Addison-Wesley, Reading, Mass., 2001.
6. *Proc. Development and Evolution of Software Architectures for Product Families,* F. van der Linden, ed., Lecture Notes in Computer Science no. 1429, Springer-Verlag, New York, 1998.
7. *Proc. Product Family Engineering,* F. van der Linden, ed., Lecture Notes in Computer Science no. 2290, Springer-Verlag, New York, 2002.
8. *Proc. Software Architectures for Product Families*, Lecture Notes in Computer Science 1951, Springer-Verlag, Berlin, 2000.

About the Author

Frank van der Linden is an international project leader at Philips Medical Systems, where he coordinates international-cooperation projects in software engineering for product families. He is also project manager for the ITEA projects Esaps and Café and was Philips' project leader of ARES (Architectural Reasoning for Embedded Systems). His main interests are in software engineering and architecture, with an emphasis on process and organization, and quality issues. He received a PhD in pure mathematics from the University of Amsterdam. He is a member of the Dutch Mathematical Society and of the Dutch Association for Theoretical Computer Science. Contact him at Philips Medical Systems, Veenpluis 4-6 5684 PC Best, Netherlands; frank.van.der.linden@philips.com.