SEI's Software Product Line Tenets

Linda M. Northrop, *Software Engineering Institute*

Software product lines are rapidly emerging as a viable and important software development paradigm. The Software Engineering Institute defines basic concepts and the activities and practices that ensure success. The author shares how-to's, success stories, and lessons learned while defining and applying this approach.

Companies, such as Hewlett-Packard, Nokia, and Motorola, are finding that using a product line approach for software can yield remarkable quantitative improvements in productivity, time to market, product quality, and customer satisfaction. This practice of building sets of related systems from common assets can also efficiently satisfy the current demand for mass customization of software. Product lines are, of course, not new in manufacturing. Boeing, Ford, Dell, and even McDonald's develop product lines. But *software product lines* are a relatively new concept. They are rapidly emerging as a practical and important software development paradigm. A product line succeeds because companies can exploit their software products' commonalities to achieve economies of production.

The Software Engineering Institute's (SEI) work has confirmed the benefits of pursuing this approach; it also found that doing so is both a technical and business decision. To succeed with software product lines, an organization must alter its technical practices, management practices, organizational structure and personnel, and business approach.

Software product lines

A software product line is a set of software-intensive systems that share a common, managed feature set satisfying a particular market segment's specific needs or mission and that are developed from a common set of *core assets* in a prescribed way.

Core assets form the basis for the software product line. Core assets often include, but are not limited to, the architecture, reusable software components, domain models, requirements statements, documentation and specifications, performance models, schedules, budgets, test plans, test cases, work plans, and process descriptions. The architecture is key among the collection of core assets.

Each system in the product line is a product in its own right. However, it is created by taking applicable components from a common asset base, tailoring them through preplanned variation mechanisms, adding new components as necessary, and assembling the collection according to the rules of a common, product-line-wide architecture.

0740-7459/02/$17.00 © 2002 IEEE

Every software product line has a predefined guide or plan that specifies the exact product-building approach.

Development is a generic term used to describe how core assets (and products) come to fruition. Software enters an organization in one of three ways: the organization builds it (from scratch or by mining legacy software), purchases it (largely unchanged, off the shelf), or commissions it (contracts with someone else to develop it especially for them). So, the term *development* might actually involve building, acquiring, purchasing, retrofitting earlier work, or any combination of these options.

Some practitioners use a different set of terms to convey essentially the same meaning. They might refer to a product line as a *product family*,[1] to the core asset set as a *platform*,[2] or to the products of the software product line as *customizations* instead of products. Others use the terms *domain* and *product line* interchangeably. We distinguish between the two. A domain is a specialized body of knowledge, an area of expertise, or a collection of related functionality. Core asset development is often referred to as *domain engineering*, and product development as *application engineering*.

Regardless of terminology, software product line practice involves strategic, large-grained reuse, which means that software product lines are as much about business practices as they are about technical practices. Using a common set of assets to build products requires planning, investment, and strategic thinking that looks beyond a single product.

Reuse, as a software strategy for decreasing development costs and improving quality, is not a new idea. However, past reuse agendas, which focused on reusing relatively small pieces of code or opportunistically cloning code designed for one system for use in another, have not been profitable. In a software product line approach, reuse is planned, enabled, and enforced. The reusable asset base includes artifacts in software development that are costly to develop from scratch.

Essential activities

Numerous organizations in various industries have reaped significant benefits using a software product line approach for their systems. Despite this diversity, we at the SEI believe we have distilled universal and essential software product line activities and practices. At the highest level of generality are three essential and highly iterative activities that blend technology and business practices. Fielding a product line involves *core asset development* and *product development* using the core assets under the aegis of technical and organizational *management*. Figure 1 illustrates this triad of essential activities.

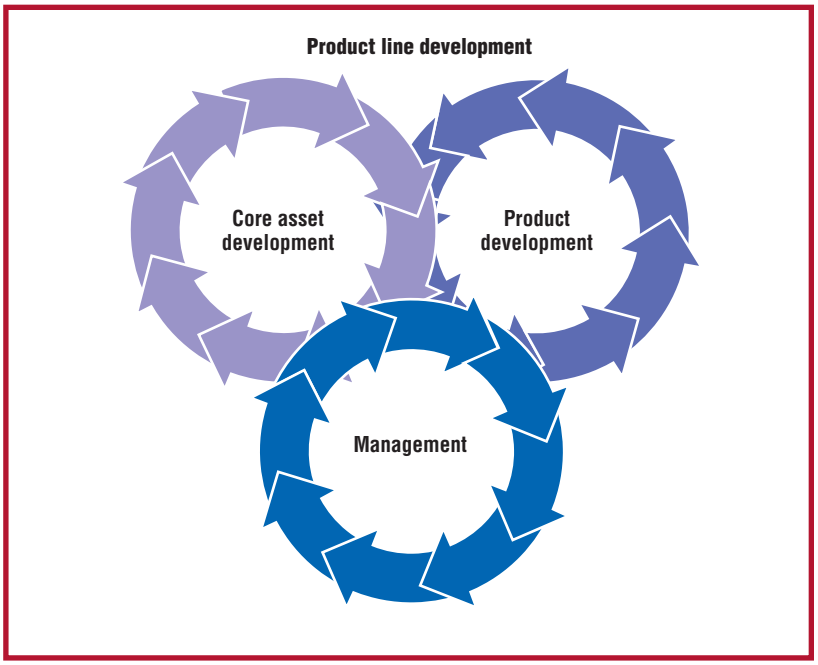


Figure 1. Essential product line activities.

The rotating arrows in Figure 1 indicate not only that companies use core assets to develop products but also that revisions of existing core assets or even new core assets might (and most often do) evolve out of product development. In some contexts, organizations mine existing products for generic assets—perhaps a requirements specification, an architecture, or software components—that they then migrate into the product line's asset base. In other cases, the core assets might be developed or procured for later use in product production. There is a strong feedback loop between the core assets and the products. Core assets are refreshed as organizations develop new products. They then track asset use, and the results are fed back to the asset development activity. Technical and organizational managers manage this process carefully at all levels.

Core asset development

The core asset development activity's goal is to establish a production capability

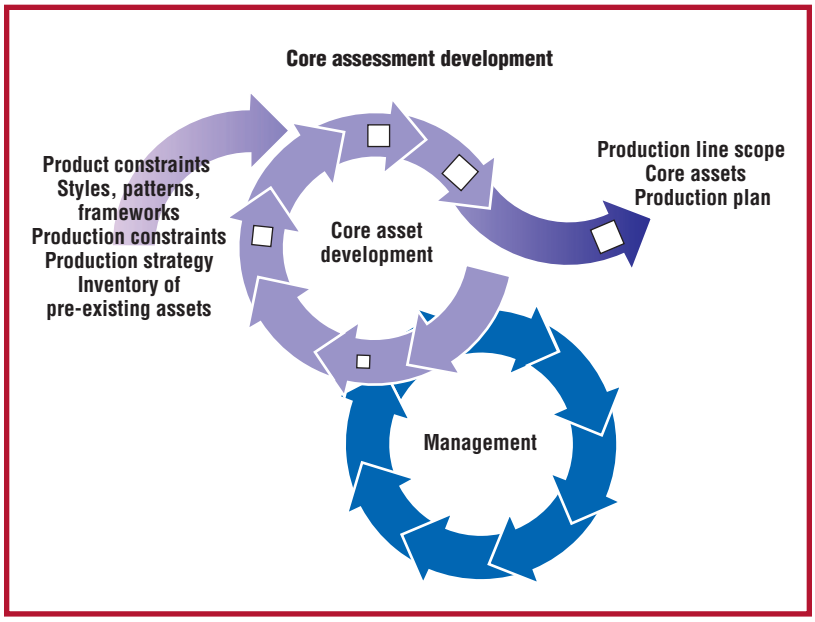


Figure 2. Core asset development.

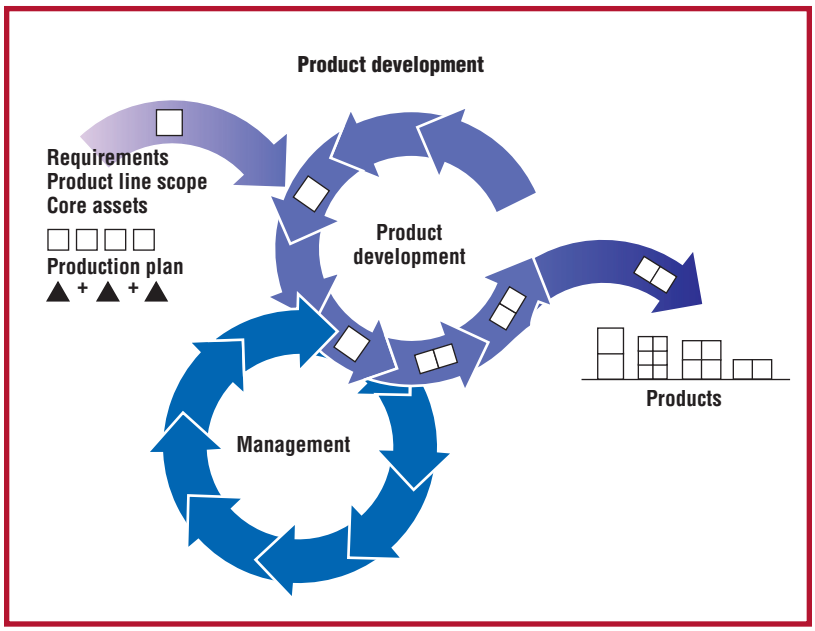
for products. Figure 2 illustrates this activity, its outputs, and necessary inputs. This activity, like its counterparts, is iterative. Its inputs and outputs affect each other. For example, slightly expanding the product line scope (an output) might admit new classes of systems to examine as possible sources of legacy assets (an input).

Inputs to core asset development include

- *Product constraints:* Commonalities and variations among the products that will constitute the product line, including their behavioral features.
- *Styles, patterns, and frameworks:* Relevant architectural building blocks that architects can apply during architecture definition toward meeting the product and production constraints.
- *Production constraints:* Commercial, military, or company-specific standards and requirements that apply to the products in the product line.
- *Production strategy:* The overall approach for realizing the core assets. This can be top down (starting with a set of core assets and spinning products off of them), bottom up (starting with a set of products and generalizing their components to produce the product line assets), or some of both.
- *Inventory of preexisting assets*: Software and organizational assets available at the outset of the product line effort that can be included in the asset base.

Besides core assets, the outputs of core asset development include a *product line scope*, which describes the products that will constitute the product line or that the product line is capable of including, and a *production plan*, which describes how products are produced from the core assets. All three outputs must be present to ensure the production capability of a software product line.

Figure 3. Product development.



Product development

In addition to the three outputs, product development activity depends on the requirements for individual products. Figure 3 illustrates these relationships; the rotating arrows indicate iteration. For example, the existence and availability of a particular product might affect a subsequent product's requirements. Creating products can have a strong feedback effect on the product line scope, core assets, production plan, and even the requirements for specific products. Product development can vary greatly depending on the assets, production plan, and organizational context.

Management

Management at the technical (or project) and organizational (or enterprise) levels must be strongly committed to the software product line effort for the product line's success. Technical management oversees the core asset development and the product develop-

ment activities, ensuring that the groups building core assets and those building products engage in the required activities, follow the processes defined for the product line, and collect data sufficient to track progress.

Organizational management must set in place the proper organizational structure that makes sense for the enterprise and ensure that organizational units receive the right resources (for example, well-trained personnel) in sufficient amounts. Organizational management determines a funding model that ensures core asset evolution and then provides the funds accordingly. It also orchestrates the technical activities in and iterations between core asset development and product development.

Management should ensure that these operations and the product line effort's communication paths are documented in an operational concept. Management mitigates risks at the organizational level that threaten a product line's success. Product lines tend to engender different relationships with an organization's customers and suppliers, and these new relationships must be introduced, nurtured, and strengthened. Management must create an adoption plan that describes the organization's desired state (that is, routinely producing products in the product lines) and a strategy for achieving that state.

Finally, someone should be designated as the product line manager and either act as or find and empower a product line champion. This champion must be a strong, visionary leader who can keep the organization squarely pointed toward the product line goals, especially when the going gets rough in the early stages.

Software product line practice areas

Beneath the surface of the three essential activities are 29 practice areas that our experience shows must be mastered for a successful product line. A *practice area* is a body of work or a collection of activities. They help make the three essential activities more achievable by defining activities that are smaller and more tractable than a broad imperative such as "Develop core assets." Most practice areas describe activities that are essential for any successful software development, not just software product lines. However, in a product line context, each takes on particular significance or must be carried out in a unique way. For example, configuration management, an identified practice area, is important for any software development effort. However, configuration management for product lines is more complex than for single systems, those developed one at a time versus using a product line approach. The core assets constitute a configuration that needs to be managed; each product in the product line constitutes a configuration that must be managed, and managing all of these configurations must be coordinated under a single process.

We have created a conceptual framework for software product line practice that provides a comprehensive description of each practice area as it relates specifically to software product line operations and the common risks associated with each.[3,4] We categorize each practice area as software engineering, technical management, or organizational management, according to the skills required to carry it out.

Software engineering practice areas

Software engineering practice areas are those that are necessary for applying the appropriate technology to create and evolve core assets and products. They are

- Architecture Definition
- Architecture Evaluation
- Component Development
- COTS Utilization
- Mining Existing Assets
- Requirements Engineering
- Software System Integration
- Testing
- Understanding Relevant Domains

Figure 4 provides a sketch of how they relate to each other.

Domain understanding feeds requirements, which drive an architecture, which specifies components. Components can be made in-house, bought on the open market, mined from legacy assets, or commissioned under contract. This choice depends on the availability of in-house talent and resources, open-market components, an exploitable legacy base, and able contractors. Their existence (or nonexistence) can affect the product line's requirements and architecture. Once available, the components must

Software engineering practice areas are those that are necessary for applying the appropriate technology to create and evolve core assets and products.

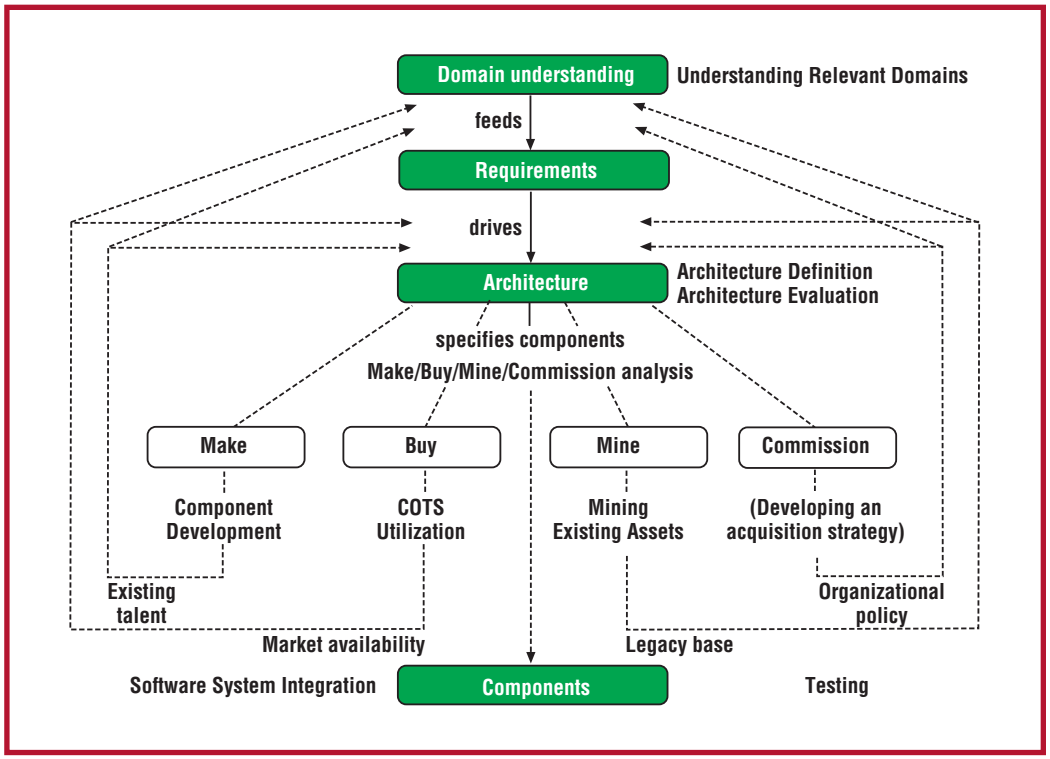


Figure 4. Relationship among software engineering practice areas.

be integrated and, along with the system, be tested. This description is a quick trip through an iterative growth cycle. It greatly oversimplifies reality but shows a good approximation of how software engineering practice areas come into play.

Technical management practice areas

Technical management practices are those that are necessary for engineering the creation and evolution of core assets and products. Technical management's practice areas are

- Configuration Management
- Data Collection, Metrics, and Tracking
- Make/Buy/Mine/Commission Analysis
- Process Definition
- Scoping
- Technical Planning
- Technical Risk Management
- Tool Support

These practices directly support and pave the way for software development activities. Scoping and Technical Planning delineate what should be built and how. Data Collection, Metrics, and Tracking and Technical Risk Management establish "health" measures for the software development efforts and help assess their current conditions. Make/Buy/Mine/Commission Analysis, Tool Support, Configuration Management, and Process Definition all contribute to a smooth development effort.

Organizational management practice areas

Organizational management practices are those that are necessary for orchestrating the entire product line effort. Practice areas in organizational management are

- Building a Business Case
- Customer Interface Management
- Developing an Acquisition Strategy
- Funding
- Launching and Institutionalizing
- Market Analysis
- Operations
- Organizational Planning
- Organizational Risk Management
- Structuring the Organization
- Technology Forecasting
- Training

Some practices, such as Building a Business Case and Funding, are required to initiate a product line approach and emphasize the business investment and planning required. Others, such as Operations and Or-

ganizational Risk Management, apply to ongoing product line efforts. Launching and Institutionalizing is about an organization's systematic growth from a given state to a higher state of product line sophistication. It is actually a context-sensitive threading of other organizational management practice areas. The sheer number of organizational management practice areas gives testament to the significant business dimension of software product lines.

Product line practice patterns

Although laying out all essential activities and practice areas has proven very helpful, an organization must still determine how to put the practice areas into play. One approach is to follow a divide-and-conquer strategy. Fortunately, although no two situations are alike, we have found that similar situations repeatedly occur. It is because of these similarities that *product line practice patterns* have emerged.[3] Patterns are a way of expressing common contexts and problem and solution pairs. They have been used effectively in many disciplines including architecture, economics, social science, and software design. For software product line practice patterns, the context is the organizational situation. The *problem* is part of the software product line effort that must be accomplished. The *solution* is the grouping of practice areas and their relations to address the problem for that context.

Following the lead of the design patterns community, we created a pattern template and have used it to define the 22 patterns (including variants) listed in Table 1.

These patterns, some of which have relationships between them, span various ranges of abstraction, scale, and purpose. For example, Factory is a composite pattern that consists of eight other patterns, so it describes the entire product line organization.

Lessons learned defining the approach

The SEI's understanding of what is involved in a software product line approach has evolved considerably. Our ideas have matured, and no doubt will continue to mature, owing to our direct involvement in software product line efforts, our discussions with others involved in product line work, and our own (sometimes heated) internal debates.

Table 1

Product line practice patterns

Pattern	Variants
Assembly Line	
Cold Start	Warm Start
Curriculum	
Each Asset	Each Asset Apprentice
	Evolve Each Asset
Essentials Coverage	
Factory	
In Motion	
Monitor	
Process	Process Improvement
Product Builder	Product Generation
Product Parts	Green Field
	Barren Field
	Plowed Field
What to Build	Analysis
	Forced March

Formulating the basic concepts

Our original thoughts were based on a domain engineering, followed by an application engineering mind-set, which had an unrealistic waterfall life-cycle mentality. We rarely encountered an organization that had the luxury of developing assets from scratch and then building products from those assets. In almost all situations, some products or assets already existed, and the asset base grew out of those. In any case, each asset continues to evolve over time. Core asset development and product development activities are highly iterative, and that iteration must be carefully managed. This latter insight led us to include management as the third essential activity.

In our original definition of software product lines, we did not prescribe how products were constructed. After much debate, we concluded that our definition was insufficiently discriminating. We added the clause "that are developed from a common set of core assets in a prescribed way." Expanding the definition proved to be an epiphany that led to other refinements. We agree with others[5] that the product line architecture plays a special role among the core assets by providing the structural prescription for products in the product lines. However, we discovered that the product line architecture alone does not provide enough

Product Line Success Stories

There is a great benefit in learning how others approached their move to product lines. We have documented four complete product line case studies.[1]

Our earliest report was a study of CelsiusTech Systems, a Swedish defense contractor supplying international navies with shipboard command and control systems.[2] Using a product line approach, they have delivered more than 50 systems from essentially the same asset base. In doing so, they have shortened delivery schedules by years, allowed a smaller staff to produce more systems, and achieved software reuse levels into the 90 percent range.

Cummins, the world's largest manufacturer of commercial diesel engines with more than 50 horsepower, made a bold move to a software product line approach for its engine control software. The results are most compelling. It previously took Cummins a year or more to bring new engine software to the test lab, but now it takes less than a week. Moreover, the product line approach lets the company augment its command of the automotive diesel engine market. It has expanded vigorously into the industrial diesel market, where just 20 software builds provide the basis for more than a thousand separate engine products; it now offers a mix of feature and platform flexibility that otherwise would require almost four times their current staff.

The US National Reconnaissance Office took advantage of commonality and built a product line asset base for its ground-based spacecraft command and control software. They commissioned Raytheon to build their asset base, the Control Channel Toolkit. The new product line's first system has seen, among other benefits, a 50-percent decrease in overall cost and schedule, and nearly tenfold reduction in development personnel and defects.

Successful product lines are also possible in small organizations, such as in Market Maker Software of Kaiserslautern, Germany, producer of Europe's most popular stock market software. Market Maker adopted a product line approach to produce an Internet version of its software. This version, which they market to other companies, must integrate with other databases and content-producing software (which run on a variety of computing platforms and servers); satisfy human-user performance requirements; and be tailored to show the exact kind of data, in exactly the kind of charts, in exactly the kind of form each customer's Web site requires. Using their software product line, it takes Market Maker as few as three days to install a tailored system for individual customers.

Others have also reported success stories: Alcatel,[3] Hewlett Packard,[4] Philips,[5] the Boeing Company,[6] and Robert Bosch GmBh[7] presented their experiences at the 2000 Software Product Line Conference (SPLC1 00).

References

1. P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley, Boston, 2001.
2. L. Brownsword and P. Clements, *A Case Study in Successful Product Line Development*, tech. report CMU/SEI-96-TR-016, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, 1996; www.sei.cmu.edu/publications/documents/96.reports/96.tr.016.html.
3. M. Coriat et al., "The SPLIT Method," *Proc. 1st Software Product Line Conf.* (SPLC1 00), Kluwer Academic Publishers, Boston, 2000, pp. 147–166.
4. P. Toft et al., "A Cooperative Model for Cross-Divisional Product Development for a Software Product Line," *Proc. 1st Software Product Line Conf.* (SPLC1 00), Kluwer Academic Publishers, Boston, 2000, pp. 111–132.
5. P. America et al., "CoPAM: A Component-Oriented Platform Architecting Method Family for Product Family Engineering," *Proc. 1st Software Product Line Conf.* (SPLC1 00), Kluwer Academic Publishers, Boston, 2000, pp. 167–180.
6. D. Sharp, "Component-Based Product Line Development of Avionics Software," *Proc. 1st Software Product Line Conf.* (SPLC1 00), Kluwer Academic Publishers, Boston, 2000, pp. 353–369.
7. S. Thiel and F. Peruzzi, "Starting a Product Line Approach for an Envisioned Market," *Proc. 1st Software Product Line Conf.* (SPLC1 00), Kluwer Academic Publishers, Boston, 2000, pp. 495–512.

prescription. Each core asset should have an associated *attached process* that specifies how to use it in product development. These attached processes get folded into what becomes the product production plan.

Another debate in the product line community was whether releases and versions of single products constituted a product line. Although others still disagree, we decided that they did not. We agree with Jan Bosch that a product line evolves over time and space.[5]

Settling on the practice area set

Being true to our technical backgrounds, we began with a greater proportion of practice areas in software engineering than in management. However, we quickly recognized the need for more management practices, and the set of technical and organizational management practices grew.

For some time, we maintained that Domain Analysis was a practice area, meaning that a formal domain analysis was required. However, CelsiusTech and other organizations with successful product lines did not conduct a domain analysis. However, they did have solid knowledge of their domains, which helped them make good product decisions. What was essential was understanding relevant domains, so that became a practice area.

Also, early on we considered understanding relevant domains, requirements engineering, and scoping as one practice area. However, we gradually found that although there were dependencies among the three, they involved different activities and players.

Determining the practice areas' contents

Experts draft individual practice area descriptions for the framework, so overlap continues to require monitoring. In reality, there are no clear boundaries between the practice areas; we could slice the effort in many different ways, but a balance, however arbitrary in some cases, is important to assert.

Some of our early ideas about software product lines were simply naïve. For example, we originally believed that the organizational structure must have two units: one to build the core assets and one to build products. Colleagues from Nokia and Hewlett-Packard, among others, pointed out that all product line development can be concentrated in a single unit, where each member is expected to be a jack-of-all-trades in the product line, doing domain engineering tasks or application engineering tasks when appropriate. Later, Bosch described four separate organizational models.[5]

Beyond practice areas

The practice area framework was (and is) an encyclopedia of software product lines,[4] but we fell short in offering concrete guidance on using that encyclopedia. There were many fits and starts about how and what to provide. We settled on product line practice patterns and have been encouraged by early positive feedback.

We have also been encouraged to connect the product line practice framework with software development standards, most especially with the Capability Maturity Model framework. We have compared the framework with the Capability Maturity Model Integration for Systems Engineering/Software Engineering V1.1.4. Although process discipline is essential for product line success, there is not one-to-one mapping between these standards. The process areas in the CMMI framework do not address 12 product line practice areas, and even for those that do cover similar subjects, the emphasis is different. More fundamentally, the product line practice framework is not a maturity model.

Lessons learned applying the approach

Besides the explicit changes in our approach, we learned these lessons:

- Product line business practices cannot be affected without explicit management commitment and involvement. We have seen too many product line efforts fail for lack of sponsorship and commitment from someone above the technical ranks.
- Organization size doesn't matter. Our original experiences were all with large organizations. Many small organizations, such as Market Maker (see the "Product Line Success Stories" sidebar), have demonstrated that they can succeed with product lines.
- Reuse has a bad reputation in many organizations owing to the failure of earlier small-grained reuse initiatives. It takes highly proactive advocacy and marketing to introduce software product lines into such cultures.
- Organizations often want an evaluation of their product line efforts. (This led us to develop the Product Line Technical Probe, a diagnostic method for examining an organization's readiness to adopt, or ability to succeed with, a software product line approach, described elsewhere.)[3]
- The lack of either an architecture focus or architecture talent can kill an otherwise promising product line effort.
- Process discipline is critical. Processes can be according to the CMM framework, Extreme Programming, or some Agile method, but they must be defined and followed. On one of our collaborations, we mistakenly introduced process improvement and software product lines simultaneously. The product line effort languished.
- The community needs more quantitative data to support software product line adoption. Moving to product lines is an investment, and decision makers want hard numbers in their business cases.

Organizations can benefit tremendously through product lines.

IEEE Software

How to Reach Us

Writers
For detailed information on submitting articles, write for our Editorial Guidelines (software@ computer.org) or access http://computer.org/software/author.htm.

Letters to the Editor
Send letters to

Editor, *IEEE Software*
10662 Los Vaqueros Circle
Los Alamitos, CA 90720
software@computer.org

Please provide an email address or daytime phone number with your letter.

On the Web
Access http://computer.org/software for information about *IEEE Software*.

Subscribe
Visit http://computer.org/subscribe.

Subscription Change of Address
Send change-of-address requests for magazine subscriptions to address.change@ieee.org.
Be sure to specify *IEEE Software*.

Membership Change of Address
Send change-of-address requests for IEEE and Computer Society membership to member.services@ieee.org.

Missing or Damaged Copies
If you are missing an issue or you received a damaged copy, contact help@computer.org.

Reprints of Articles
For price information or to order reprints, send email to software@computer.org or fax +1 714 821 4010.

Reprint Permission
To obtain permission to reprint an article, contact William Hagen, IEEE Copyrights and Trademarks Manager, at whagen@ieee.org.

Software product lines epitomize the concept of strategic, planned reuse, and differ from the opportunistic reuse of the past that has been largely discredited. Organizations can benefit tremendously through product lines. A number of global software trends make product lines more doable today than in the past, such as rapidly maturing and increasingly sophisticated software development technologies, mature object technology, vendor-available components with tremendous functional capability, increased realization of the importance of architecture, universal recognition of the need for process discipline, product line case studies, workshops, and education programs. Nevertheless, there are needs in many areas. For example, better product line tool support and more supportive business models and data are imperative. However, the industry trend toward software product lines seems indisputable. The SEI believes that software product lines are here to stay.

Acknowledgments

Many people have contributed to the SEI's product line work, both inside and outside the SEI. Although Paul Clements and I have led the charge, the entire SEI Product Line Practice Initiative team has contributed to the basic concept evolution, to the focusing of the key ideas, and to the work's conceptual integrity. Much of the information that the SEI has assimilated has come from software community members who have built software product lines, sometimes with our help. They have graciously participated in our conference and software product line workshops and shared with us their knowledge and experience.

References

1. P. America et al., "CoPAM: A Component-Oriented Platform Architecting Method Family for Product Family Engineering," *Proc. 1st Software Product Line Conf.* (SPLC1 00), Kluwer Academic Publishers, Boston, 2000, pp. 167–180.
2. P. Toft et al., "A Cooperative Model for Cross-Divisional Product Development for a Software Product Line," *Proc. 1st Software Product Line Conf.* (SPLC1 00), Kluwer Academic Publishers, Boston, 2000, pp. 111–132.
3. P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley, Boston, 2001.
4. P. Clements and L. Northrop, "A Framework for Software Product Line Practice," 2000; www.sei.cmu.edu/plp/framework.html.
5. J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*, Addison-Wesley, Boston, 2000.

Linda M. Northrop's biography appears on page 27.

For more information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.