Carnegie Mellon
Software Engineering Institute

SEI interactive

Volume 2 . Issue 3 . September 1999

Full Issue

http://interactive.sei.cmu.edu

S E I i n t e r a c t i v e

SEI Interactive

Volume 2 . Issue 3 . September 1999

About the SEI

Mission

The SEI mission is to provide leadership in advancing the state of the practice of software engineering to improve the quality of systems that depend on software.

The SEI expects to accomplish this mission by promoting the evolution of software engineering from an ad hoc, labor-intensive activity to a discipline that is well managed and supported by technology.

SEI Work

The SEI program of work is grouped into two principal areas:

- Software Engineering Management Practices
- Software Engineering Technical Practices

Within these broad areas of work, the SEI has defined specific initiatives that address pervasive and significant issues impeding the ability of organizations to acquire, build, and evolve software-intensive systems predictably on time, within expected cost, and with expected functionality. Visit the initiative page on the SEI Web site [http://www.sei.cmu.edu] for more information.

From the Director

Product Lines: High-Quality Software Really Fast

Stephen E. Cross

A New Software Paradigm

The SEI's Paul Clements calls product line practice "a new paradigm for the new century." I could not agree more. As Clements points out, "If the pitfalls are successfully negotiated, the result is an enviable capacity to deliver extremely large systems on time and within budget." Such a powerful approach can provide enormous benefits to software engineers working on Department of Defense contracts or commercial products.

The product line approach is in essence the strategic reuse of all of an organization's core assets to support the rapid development of similar systems from core assets. I say strategic because the decision to employ the product line approach is both a business and technical decision, as described in this issue. Core assets include not only reusable software, but the system's architecture, proven processes related to the development and testing of systems, and all the knowledge related to the development of systems (for example, test plans, requirement documents, etc.).

The organization that successfully employs product line approaches invests wisely in its core assets. It invests in its people and puts its best people on the maintenance—the stewardship—of its core assets. For example, over time the quality of architecture and software-based core assets improves. This means that the overall quality of a new system assembled from that architecture and those software components is of higher quality. Experience has shown that much less time is then required in the system-integration and test phases. Hence, the product line approach is a "product, process, and people" improvement strategy for reusing an organization's core assets for delivering high-quality software with reduced cycle times.

Product Lines and the SEI's Work

Much of the work of the SEI supports organizations that seek to employ a product line approach. Besides the SEI's Product Line Practice (PLP) initiative, consider the following:

- Architecture Tradeoff Analysis (ATA). The software architecture is a critical core asset, the "blueprint" that embodies the critical system attributes (for example, performance, security, reliability, modifiability, etc.) The ATA initiative provides guidance on how to analyze an architecture. Conducting an exercise that is part of the architecture tradeoff analysis method (ATAM) typically consumes three calendar days. Not a bad price for the peace of mind you get by knowing that your architecture is going to meet the needs of your product line.

- Process improvement. Turning out a product line requires a certain maturity of process. People have to be disciplined. Product lines are about following established channels of communication and feedback, especially between the core asset developers and the consumers of those core assets—namely the product-producing projects. In our experience, process improvement and the migration to product line strategy go hand-in-hand, especially for those organizations with low process maturity.

- Component-based systems. Product lines often epitomize component-based systems. A product line architecture is built to accommodate variation and extension, and one mechanism for achieving this is the ability to replace components quickly and easily in order to achieve the special variations needed for different products.

The two Spotlight articles in this issue highlight some of the ways in which product line practices cut across all areas of software development by focusing on the activities at four companies with successful product line programs. Our Roundtable discussion provides additional organizational contexts through a series of scenarios.

First Product Line Conference

To help further the widespread use of product line practices, we at the Software Engineering Institute will sponsor the first Software Product Line Conference (SPLC1) in Denver, Aug. 28–31, 2000. The conference will be open to the software community at large, but the focus will be on achievements that are triggering the growing adoption of product line practices.

We invite researchers and practitioners to contribute to SPLC1 by submitting refereed archival technical papers, topical panels, tutorials, and/or workshops. The deadline for submission of papers and proposals is Dec. 15, 1999. We hope you will join us for this unique opportunity to exchange ideas and experiences related to software product lines and to broaden the community interest in product line technology. For further information, please visit the SPLC1 Web site at

http://www.sei.cmu.edu/plp/conf/SPLC.html

Improving Software in General

Product lines are, of course, just one area where the SEI is improving the practice of software engineering. I am often asked questions such as: “What are you going to do about bad software?” In fact, Martha Heller asked that very question in her recent

Sound Off column in *CIO* magazine (available online at http://comment.cio.com/070699_sound.html). I told her: “Better software is a joint responsibility of the buyer and the builder,” and I advised that dissatisfied CIOs adopt the best practices in the SEI’s Software Acquisition Capability Maturity Model® (SW-CMM®). She took my comment a step further, proposing as one solution that CIOs “could get together and refuse to upgrade unless the software adheres to management practices developed by quality experts like Cross and his team at the SEI.”

Indeed, the SEI is currently working on ways to motivate or provide incentives for organizations to use best practices, such as product line approaches, the CMM Integration product suite (which embodies the practices described in the SW-CMM® v 2c), the Personal Software Process℠, and other best practices. One way to provide incentives would be the use of model contracting language that requires software developers to adhere to these practices.

We expect to soon release some of these ideas to the developer community. Look to future issues of *SEI Interactive* for more on what is likely to be a very interesting debate.

From the Editor

Welcome to SEI Interactive

Welcome to the sixth installment of *SEI Interactive*.

One of the interesting tasks that comes with editing an online magazine is perusing our anonymous Web statistics to try to glean information about our readers and their approaches to reading *SEI Interactive*.

Sometimes the effort raises questions rather than answering them. Why, for example, is our readership highest on Tuesdays? And why is 2:00-4:00 p.m. our busiest time of day? Why, this past August, did we have three times as many requests for pages—or “hits”—from readers in Finland as we had from readers in Norway?

For the most part, however, the Web statistics give us exciting and encouraging insights. For example:

- Our readership has grown steadily since the launch of *SEI Interactive* with the June 1998 issue. Since then, our readership has grown by 500 percent, and each new release brings another large group of new readers. (We measure this by the statistic “distinct hosts served,” meaning the number of unique IP addresses that access the *SEI Interactive* server in a month.)

- We have readers in approximately 60 countries. After the United States, the countries with the highest readership are Germany, Canada, the Netherlands, Australia, the United Kingdom, and Japan.

- Every item in *SEI Interactive* has a long shelf life. Our Archives section is truly a rich repository of information from which the software engineering community frequently draws. Indeed, during some months, our archived articles and columns receive almost as many hits as our current material.

- Our columnists have strong readership, led by Watts Humphrey and his “Watts New” column, which is consistently one of the five most requested pages. The columns also generate the strongest interaction with our discussion groups feature—and we’re happy to say that our columnists themselves actively participate.

- Readers use *SEI Interactive* as an on-ramp to other SEI information. Our list of recent SEI publications is consistently the third- or fourth-most-requested file, and close behind is the “Announcements” page. Clearly, *SEI Interactive* is a favorite method for finding out about the SEI in general.

With this issue we want to alert you to a new feature. In response to requests from readers, we have gathered all the content of this issue into one PDF file. So, in addition to getting a PDF version of each article, you can also get the entire issue, with its own cover and table of contents.

Bill Thomas
SEI Interactive editor-in-chief
Software Engineering Institute

The SEI Interactive Team:

Mark Paat, communication design
Bill McSteen
Jamie Teasdale
Barbara White

Thanks also to Paul Clements, the guest editor for this issue's Features section, and to all of our content reviewers, including Steve Cross, John Goodenough, and Linda Northrop.

Introduction

Product Line Practice: An Effort Worth Making

“Software developed as a product line promises to be a dominant development paradigm for the new century,” according to Paul Clements of the SEI’s Product Line Practice Initiative. But the challenges involved cannot be taken lightly. “The successful transition to product-line technology requires a careful blend of technology, process, organization, and business factors improvement,” he says, but adds that the payoff is worth it: “If the pitfalls are successfully negotiated, the result is an enviable capacity to deliver extremely large systems on time and within budget.”

This issue of *SEI Interactive* examines in depth the high-stakes subject of product line practice.

Our Background article, “A Framework for Software Product Line Practice,” provides a slightly condensed version of the first two chapters of version 2.0 of this framework, which was released by the SEI in July 1999. The framework is intended to be a living document that will aid the software development and acquisition communities. Each version represents an incremental attempt to capture information about successful product line practices. This information has been gleaned from studies of organizations that have built product lines, from direct collaborations on software product lines with the SEI’s customer organizations, and from leading practitioners in software product lines.

In our first of two Spotlight articles, “Software Product Lines: A New Paradigm for the New Century,” the SEI’s Paul Clements discusses the advantages of product lines, uncovers some of their pitfalls, and shows by example the kinds of successes that organizations can enjoy. Our second Spotlight article presents further examples, describing how product line programs have been handled at Cummins Engine, Raytheon, and Hewlett-Packard. Those companies have all enjoyed substantial reductions in time to market, cost, and risk, and significant gains in efficiency and quality. But the leaders of those companies’ programs are quick to point out that large-scale technical and cultural changes are required. They share some of the lessons that they have learned.

This issue’s Roundtable captures a panel discussion from the 1999 Software Engineering Symposium. The panelists, all members of the SEI technical staff, depict a specific usage scenarios for the latest version of the product line practice framework given a specific organizational context, and illustrating the use of practices in the areas of launching a product line, scoping a product line, using commercial off-the-shelf software in a product line, and defining a product line architecture.

Finally, our Links feature offers a guided tour of information available on the Web about product line practice.

Background

A Framework for Software Product Line Practice

Paul Clements, Linda M. Northrop

This article is excerpted from the first two chapters of *A Framework for Software Product Line Practice, Version 2.0.* The framework is intended to be a living document that will aid the software development and acquisition communities. Each version represents an incremental attempt to capture information about successful product line practices. This information has been gleaned from studies of organizations that have built product lines, from direct collaborations on software product lines with customer organizations, and from leading practitioners in software product lines. In the full document, available on the Web at http://www.sei.cmu.edu/plp/framework.html, Chapter 3 provides a detailed description of how product line practices could be applied to software engineering, technical management, and organizational management practice areas. “Not all of the practice areas have been defined, but our goal is to release the framework in increments to get the information out sooner and to get feedback and contributions,” writes co-author Linda Northrop, director of the Product Line Systems Program at the SEI. “Future versions will build upon the current foundation by completing still other practice area descriptions, and by describing a small number of product line scenarios involving the development, acquisition, and/or evolution of a software product line.” Northrop requests that readers provide feedback and make contributions to the framework by contacting her at lmn@sei.cmu.edu.

Introduction

Software product lines are emerging as a new and important software development paradigm. Companies are finding that the practice of building sets of related systems from common assets can yield remarkable quantitative improvements in productivity, time to market, product quality, and customer satisfaction. Organizations that acquire, as opposed to build, software systems are finding that commissioning a set of related systems as a commonly developed product line yields economies in delivery time, cost, simplified training, and streamlined acquisition. But along with the gains come risks. Although the technical issues in product lines are formidable, they are but one part of the entire picture. Organizational and management issues constitute obstacles that are at least as critical to overcome, and may in fact add more risk because they are less obvious.

Building a software product line and bringing it to market requires a blend of skillful engineering as well as both technical and organizational management. Acquiring a software product line also requires this same blend of skills to position the user organizations to effectively exploit the commonality of the incoming products, as well as to lend sound technical oversight and monitoring to the development effort. These skills are necessary to overcome the pitfalls that may bring disaster to an unsophisticated organization.

Organizations that have succeeded with product lines vary widely in

- the nature of their products
- their market or mission
- their organizational structure
- their culture and policies
- their software process maturity
- the maturity and extent of their legacy artifacts

Nevertheless, there are universal essential activities and practices that emerge, having to do with the ability to construct new products from a set of core assets while working under the constraints of various organizational contexts and starting points.

Every organization is different and comes to the product line approach with different goals, missions, assets, and requirements. Practices for a product line developer will be different from those for a product line acquirer, and different still for a component vendor. Appropriate practices will vary according to

- the type of system being built
- the depth of domain experience
- the legacy assets on hand
- the organizational goals
- the maturity of artifacts and processes
- the skill level of the personnel available
- many other factors

There is no one correct set of practices for every organization, but this document contains practices that we have seen work successfully in practice.

What is a Software Product Line?

A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission.

Substantial economies can be achieved when the systems in a software product line are developed from a common set of core assets, in contrast to being developed one at a time in separate efforts. Using common assets, a new product is formed by taking applicable components from the asset base, tailoring them as necessary through pre-planned variation mechanisms such as parameterization, adding any new components that may be necessary, and assembling the collection under the umbrella of a common, product line-wide architecture. Building a new product (or system) becomes more a matter of generation than creation; the predominant activity is integration rather than programming.

Organizations routinely produce new releases and versions of products. Each of these new versions and releases is typically constructed using the architecture, components, test plans, etc. from the prior releases. Why are product lines different? Two elements of product lines capture the essence of the answer: the production of a related set of products and their production from a core asset base. The production itself is specified in a production plan.

Within a product line, an organization has multiple products, each of which is going through its own cycles of release and version simultaneously. Thus, the evolution of a single product must be considered within a broader context, i.e. the evolution of the product line as a whole.

The second essential aspect of the definition is the production of instances from a core asset base. Thus, we are concerned in this document with the means of production and the structure of the producing organization. How a customer might view a collection of products and interact with the producers is only of ancillary interest. It is difficult to discern from an examination of the behavior of a system (or a collection of systems) whether they were constructed from a core asset base. In fact, most customers care only about price, schedule, function, and quality. Our focus is on what it means for a producer to be producing, or for an acquirer to be acquiring, multiple products from the same asset base simultaneously.

Benefits and Costs of a Product Line

A product line epitomizes strategic reuse. Core assets extend far beyond mere code reuse. Each product in the product line can be turned out by taking advantage of analysis, design, code, testing, planning, training, and a host of other activities that have already been performed for previous products in the product line. For each reuse benefit, however, there is usually an associated cost and a caveat to achieving the benefit. The items that affect both the product line and new products as well as their benefits and costs are shown in the following table:

Asset	**Benefit**	**Costs**
Architecture, architecture specification, architecture evaluation: The architecture for the product line is the blueprint for how each product is assembled from the components in the asset base. The right architecture provides for all of the quality attributes of the products; the wrong architecture precludes achieving desired quality.	Architecture represents a significant design investment by the organization's most talented engineers. Leveraging this investment across all products in the product line means that for subsequent products, the most important design step is completed.	The architecture must support the variation inherent in the product line, which imposes an additional constraint on it.
Software components: The software components that populate the asset base form the building blocks for each product in the product line.	The design decisions, data structures, algorithms, documentation, reviews, code, and debugging effort can all be leveraged across all products in the product line.	The components must be designed to be robust and applicable across a wide range of product contexts, possibly complicating their design. Often, components must be designed to be more general without loss of performance. Variation points must be built in.
Performance modeling and analysis: For products that must meet real-time constraints, analysis must be performed to show that the system's performance will be adequate.	A new product can be fielded with high confidence that real-time and distributed-systems problems have already been worked out because the analysis and modeling can be reused from product to product. Process scheduling, network traffic loads, deadlock elimination, data consistency problems, and the like will all have been modeled and analyzed.	Reusing the analysis may impose constraints on moving processes among processors, on the creation of new processes, or on synchronization among existing processes.
Tools and processes for software development, and process for making changes: The infrastructure for turning out a software product requires a substantial investment.	Configuration control boards, configuration management tools and procedures, management processes, and the overall software development process are in place and have been used before. Tools and environments purchased for one product can be amortized across the entire product line.	The boards, tools, and procedures must be more robust to account for the differences between managing a product line and managing a single product.
People, skills, training: In a product line organization, the development staff works on the entire product line, not just a single product (although some people will work on a single product at a time).	Because of the commonality of the applications, personnel can be transferred among projects as required. Their expertise is applicable across the entire product line. Their productivity, when measured by the number of products to which their work applies, rises dramatically.	Personnel must be trained beyond general software engineering and corporate procedures to ensure that they understand and can use the assets and procedures associated with the product line. New personnel must be much more specifically trained for the product line. Training materials must be created that address the product line. As product lines mature, the skills required in an organization tend to change, away from programming and systems engineering and toward relevant domain expertise and technology

		forecasting. This transition must be managed.

Table 1: Costs and Benefits of Product Lines

For each of these assets, the investment cost is usually much less than the benefit. Also, most of the costs are one-time costs but the benefits accrue with each new product release. However, an organization that attempts to institute a product line without being aware of the costs is likely to abandon the product line concept before seeing it through.

In workshops on product line practice conducted by the Software Engineering Institute, workshop participants shared some of the specific quantified benefits of product line practice. These included the following:

- being able to use one person to handle the integration and testing of a 1.5M SLOC Ada real-time safety-critical shipboard command and control system
- increasing productivity (as measured by feature density per shipped product per engineer per unit time) six-fold over a period of three years
- building a software system capable of running a new diesel engine over a weekend, as opposed to the full year that it used to take
- being able to join a military simulation exercise 12 months ahead of the schedule predicted without asset reuse

These workshops have also revealed examples of the costs:

- canceling three large projects to devote resources to building the base of core assets
- reassigning staff who could not adjust to the product line way of doing business
- suspending product output for a year while the new practices were put into place

Companies who bore these costs and made the successful transition to product line practice all agree that the payoff more than compensated for the effort, but these costs underscore the point that product line practice is often uncharted territory. A framework such as this one can provide the necessary insights and guidance.

Starting Versus Running a Product Line

Many of the practice areas in this framework are written from the point of view of describing an in-place product line capability. Of course, we recognize that the framework will be used to help an organization put that capability in place, and ramping up to a product line is in many ways different from running one on a day-to-day basis.

We felt it was important to describe the end or "steady state" so that readers could understand the goals. However, to address the issues of starting (rather than running) a product line shop, the reader is referred to the "Launching and Institutionalizing a Product Line" practice area in the full document. (See http://www.sei.cmu.edu/activities/plp/framework.html)

Product Line Essential Activities

At its essence, fielding a product line involves *core asset development* or *acquisition*, and *product development* or *acquisition* using core assets. These two activities can occur in either order (new products are built from core assets, or core assets are extracted from existing products). Often, products and core assets are built in concert with each other. Core asset development has been traditionally called domain engineering. Product development from core assets is often called application engineering. The entire effort is staffed, orchestrated, tracked, and coordinated by management. The following figure illustrates this triad of essential activities. The iteration symbol at the center represents the decision processes that coordinate the activities.

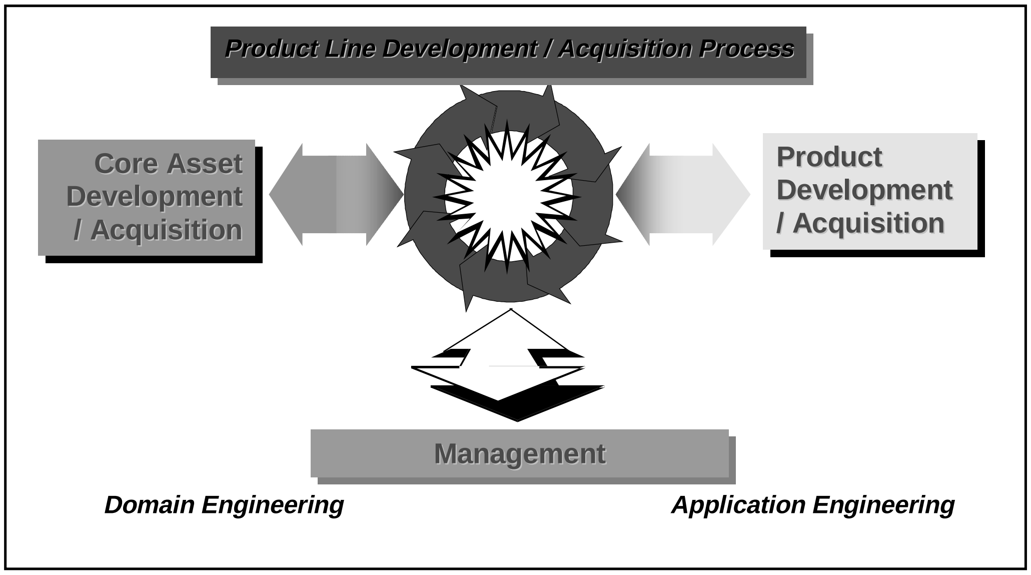

Product Line Development / Acquisition Process
Core Asset Development / Acquisition
Product Development / Acquisition
Management
Domain Engineering
Application Engineering

Figure 1: Essential Product Line Activities

The bi-directional arrows indicate not only that core assets are used to develop products, but that revisions to or even new core assets might, and most often do, evolve out of product development. The diagram is neutral about which part of the effort is launched first. In some contexts, already-existing products are mined for generic assets—a requirements specification, an architecture, software components, etc.—that are then migrated into the product line's asset base. In other cases, the core assets may be developed or procured first to produce a set of products that is merely envisioned and does not yet exist.

There is a strong feedback loop between the *core assets* and *products*. *Core assets* are refreshed as new products are developed. In addition, the value of the *core assets* is realized through the *products* that are developed from them. As a result, the *core assets* are made more generic by considering potential new *products* on the horizon. There is a constant need for strong, visionary management to invest resources into the development of the *core assets*. Management must also precipitate the cultural change to view new *products* in the context of the available assets. New *products* must either align with the existing assets, or the assets must be updated to reflect the new *products* that are being marketed. Both the *core asset development and acquisition* and *the product development or acquisition* are themselves iterative in nature as illustrated in the following figure. Iteration is inherent in product line activities, in turning out core assets, in turning out products, and in the coordination of the two.

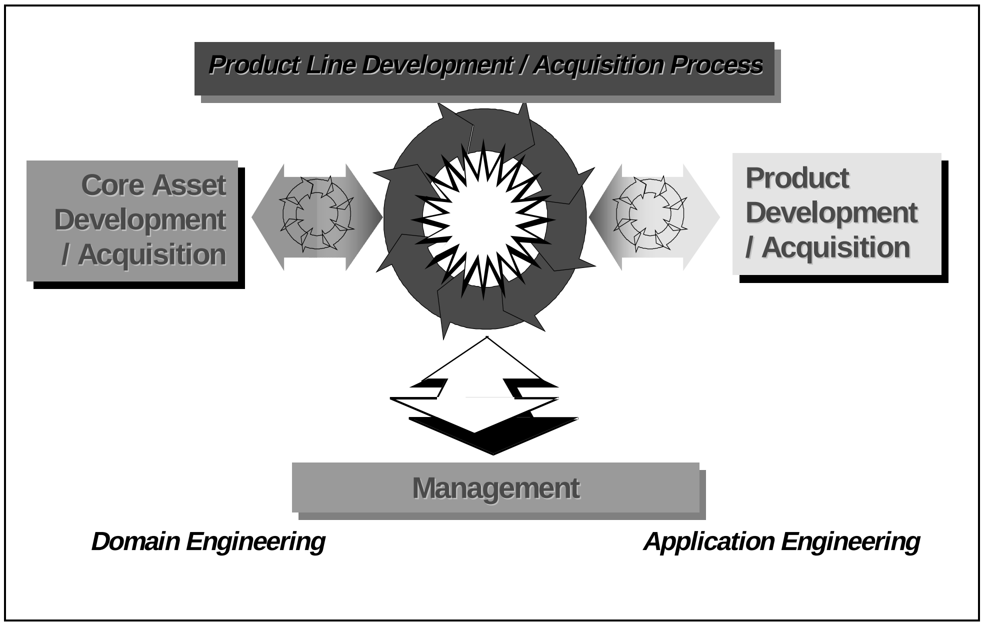

Product Line Development / Acquisition Process
Core Asset Development / Acquisition
Product Development / Acquisition
Management
Domain Engineering
Application Engineering

Figure 2: Iteration in Product Line Activities

Core Asset Development and/or Acquisition

The goal of the core asset activity is to produce or to acquire a product production capability, which requires the following three things:

1. *Product space.* A product space is a description of the initial products that constitute the product line. This description is typically cast in terms of the things that the products all have in common, and the ways in which they vary from one another (as opposed to an enumerated list of product names). This output is a description of the *space* of products that the product line is capable of including. It expresses the product line's *scope*, which is discussed below. Of course, the product space evolves as market conditions change, as the organization's plans change, or as new opportunities arise. Evolving the product space is the starting point for evolving the product line to keep it current.
2. *Core assets.* Core assets are the basis for production of products in the product line. These assets almost certainly include an architecture that the products in the product line will share, as well as software components that are designed (or reengineered from existing systems) for systematic reuse across the product line. Software components also bring with them test plans, test cases, integration plans, and all manner of design documentation. Other assets, as mentioned previously, also populate this set. Commercial off-the-shelf (COTS) components, if adopted, also constitute core assets. Although every core asset will not necessarily be used in every product in the product line, all are used in enough of the products to make their coordinated development, maintenance, and evolution pay off.
3. *Production plan.* A production plan describes how the products are produced from the core assets.

These three outputs feed the product development or acquisition activity, which turns out products that serve a particular customer or market niche. The products are built with the core assets using the production plan.

Inputs to the development and acquisition of core assets include the following:

1. *Product constraints:* What are the commonalities and variations among the products that will constitute the product line? What are their behavioral and quality requirements? What features do the market and technology forecasts say will be beneficial in the future?
2. *Production constraints:* What commercial, military, or company-specific standards apply to the products in the product line? Is there an underlying infrastructure that these products must be built on top of? What are the time-to-market or time-to-initial-operating-capability requirements for each? What off-the-shelf components should be used? Which legacy components could/should be reused?
3. *Styles, patterns, and frameworks:* What are the relevant architectural building blocks that can be applied toward meeting the product and production constraints?
4. *Production strategy:* Will the product line be built from the top down (starting with a set of core assets and spinning off products from those) or bottom up (starting with a set of products and generalizing their components to produce the product

line assets)? What will the transfer pricing strategy be (i.e., how will the cost of producing the generic components be divided among the cost centers for the products)? Will generic components be produced internally or purchased on the open market? Will products be automatically generated from the assets or will they be assembled?

5. *Inventory of pre-existing assets:* What are the software and organizational assets available at the outset of the product line effort? Are there libraries, frameworks, algorithms, tools, and components that should be utilized?

The following figure illustrates the process of developing or acquiring the core asset base and product line production capability. This activity is iterative, as suggested by the iteration symbol in the center. Again, the arrows are double-headed to suggest that there is no one-way causal relationship from inputs to outputs; rather, the inputs and outputs of this activity affect each other. For example, slightly expanding the *product space* may admit whole new classes of systems to examine as possible sources of legacy assets. Similarly, a *production constraint* (such as mandating the use of CORBA) may lead to restrictions on the *architectural styles and patterns* that will be considered for the product line as a whole (such as the message-passing distributed object style). This restriction, in turn, will determine which pre-existing assets are candidates for reuse or mining.

Defining the Product Space

Defining the product space is a matter of determining the scope of, or *scoping* the product line. The scope of the product line determines how many products the product line comprises. The scope defines the commonality that every member shares and the ways in which they vary from each other. For a product line to be successful, its scope must be carefully defined. If product members vary too widely, then the core assets will be strained beyond their ability to accommodate the variability, economies of production will be lost, and the product line will collapse into the old-style one-at-a-time product development effort. If the scope is too small, then the core assets might not be built in a generic enough fashion to accommodate future growth, and the product line will stagnate.

Further, the scope of the product line must target the right products, as determined by prevailing or predicted market factors, the nature of competing efforts, or the organization’s business goals for embarking on a product line approach (such as merging a set of similar but currently independent product development projects).

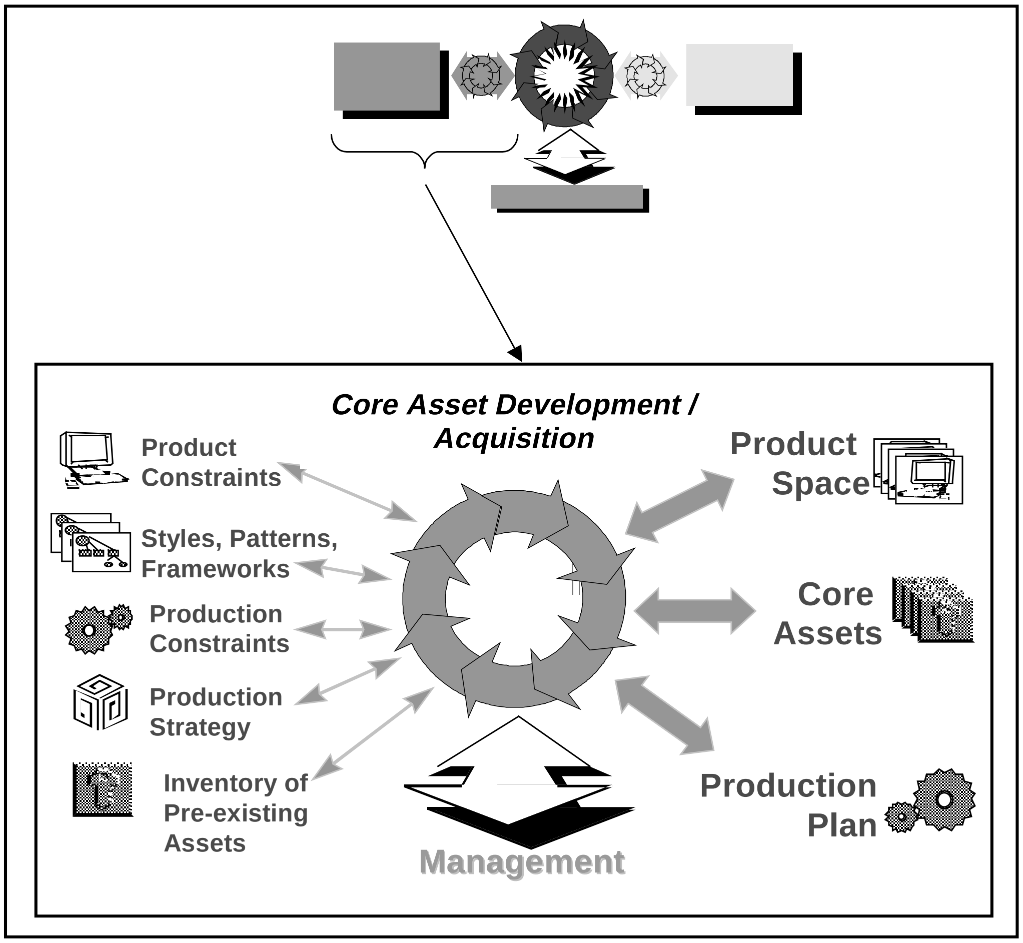


Figure 3: Core Asset Development/Acquisition

Membership in the product line can serve more than one purpose. Perhaps an organization is building or acquiring several systems that are similar to each other but do not take advantage of that similarity. They wish to merge the efforts to gain economies of scope, in which case membership is defined by an enumerated list of products. Perhaps the organization is aiming to capture or penetrate a market segment; they wish to establish a flexible, quick-response capability for launching new products in that market. In this case, membership is a function of marketing projections and is defined to include not only an initial set of products but also an abstract set of products that have not yet been built or completely defined, but the possibility is being considered or planned. In most cases, the scoping must continue after the initial scope has been defined; new market opportunities may arise, or new opportunities for strategic reuse and merging of projects may make themselves known.

Scoping the product line must account for any existing product constraints, such as a set of computing platforms on which the products must run or the set of features that the products must provide. Knowledge of similar products or systems is essential. Marketing and technology forecasts are used to determine what features the product set should make available both now and in the future.

Practice areas relevant to determining the product space include the following:

- product line scoping
- domain analysis
- market analysis
- requirements, elicitation, analysis, and management
- technology forecasting

Producing the Core Assets

Core assets for a software product line include the system and software architectures[1] that all of the products will share, and more tangible assets such as software components and their supporting artifacts.

Architecture is a critical output of the core asset activity. The architect's responsibility includes choosing (or crafting) the architecture that will satisfy the needs of the product line in general and the individual products in particular. Equally important, the architect must communicate the architecture to those who will be building core assets (software components, supporting documentation, etc.) and those who will be building products. The architecture defines those software components that are candidates to become core assets. Conformance rules must be put in place for ensuring that the products in the product line conform to the architecture; that is, that no product begins to go its own way and depart from the overall architectural scheme. The architect is also responsible for ensuring that the architecture remains viable over the life of the product line; this responsibility may require technology forecasting.

The architect must account for the intended scope of the product line before the architect can produce an architecture to satisfy it. The commonality defined by the scope will tend to become embedded in the architecture; the variability in the scope will tend to become embedded in the tailorable or replaceable components.

[1] Software architecture is the structure or structures of the system, which comprise software components, the externally visible properties of these components, and the relationships among them [Bass 98a].

To produce an architecture for a product line requires three main elements:

1. *Product space.* This is the product space that defines the product line scope (although, in fact, the architecture may influence the product space).
2. *Relevant styles, patterns and frameworks.* Architectures these days are seldom built from scratch; rather, they evolve from previously-applied solutions to similar problems. Architectural styles represent a current approach to re-using architectural design solutions. Architectural style catalogs exist that explain the properties of a particular style, including how well-suited each is for achieving specific quality attributes such as security or high performance [Bass 98a]. Patterns occupy the same role at a finer granularity of design. Whereas architectures prescribe how large-grained components (subsystems) interact with each other, patterns usually suggest ways to implement individual (or groups of finer-grained) components.
3. *Knowledge of legacy systems.* Components may be mined from legacy systems. Such components become prime candidates for components in the core asset base.

Of course, an architecture is but one of the core assets that constitute the reusable portion of a product line. These assets include (but are not limited to) those artifacts associated with reusable software components: requirements specifications, design/interface specifications, code, test plans/cases/procedures, performance engineering models, review forms and procedures, and so forth.

Component production or acquisition requires the architecture, which defines the components that will comprise each product. The architecture will also determine which components are common across all products (or at least across subsets of the product line) and define the necessary variations among instances of those components. Also required is an inventory of pre-existing assets, including commercial off-the-shelf software, to help determine whether components are to be reengineered, purchased, or built afresh.

Finally, part of creating the core asset base is defining how that core asset base will be updated as the product line evolves, as more resources become available, as fielded products are maintained, and as technological changes or market transitions affect the product scope.

Practice areas that are relevant to producing the product line architecture, core components, and other core assets include the following:

- architecture exploration and definition
- architecture evaluation
- requirements elicitation, analysis, and management

- COTS utilization
- make/buy/mine/outsource analysis
- mining existing assets
- process modeling and implementation
- software systems integration
- technical risk management
- component development
- testing
- tool support
- developing and implementing an acquisition strategy

Developing the Production Plan

Successful product line practice depends on a well-understood, effective set of practices and procedures for building products, as well as for building and evolving the architecture, the other core assets, and the individual products. The overall integrity of the product line construction and maintenance steps is critical. Some organizations maintain a common set of development tools or environments. The production plan takes these and other considerations into account. The production plan has to determine whether the product line will be built from the top down (starting with a set of core assets and spinning off products from those) or bottom up (starting with a set of products and generalizing their components to produce the product line assets). Moreover, each of the core assets will likely have an attached process that prescribes its tailorability and use in product development. The production plan is the overarching scheme that links these individual processes.

Another part of the production plan is the definition of metrics to measure organizational improvement as a result of the product line (or other process improvement) practices, and plan for collecting the data to feed those metrics.

Finally, a major part of the production plan is the Product Line Concept of Operations, which defines the product line organization and its responsibilities and processes.

For the production plan to be put in place, the following information is necessary:

- The organization’s business goals for moving to a product line approach must be understood. Organizational goals must first be known so that appropriate measures can be determined, taken, and tracked to learn whether the goals are being achieved.

- Production constraints must be identified. What commercial-, military-, or company-specific standards apply to the products in the product line? What are the time-to-market or time-to-initial-operating-capability requirements for each? What off-the-shelf components should be used? What legacy components could or should be reused? Production constraints catalog the requirements for producing products in the product line. They include the use of enterprise standards, in-house software development environments, and requirements for how expensive it is (in terms of resources used or time it takes) to produce a product in the product line.

Practice areas relevant to producing the production plan include the following:

- data collection, metrics, and tracking
- process modeling and implementation
- planning and tracking
- configuration management
- operations
- technical risk management
- developing and implementing an acquisition strategy
- tool support
- software system integration

Product Development or Acquisition

The activity of turning out products is the ultimate product line goal; core asset development is only one means toward that end. The product development/acquisition activity depends on the three outputs described above—the product space, the core asset list, and the production plan—plus the requirements for individual products. The following figure illustrates these relationships. Once more, the arrows are double-headed to indicate intricate relationships, and the circular arrow chain represents iteration. For example, the existence and availability of a particular product may well affect the requirements for a subsequent product. As another example, building a product that has previously unrecognized commonality with another product already in the product line will create pressure to update the core assets and provide a basis to exploit that commonality for future products.

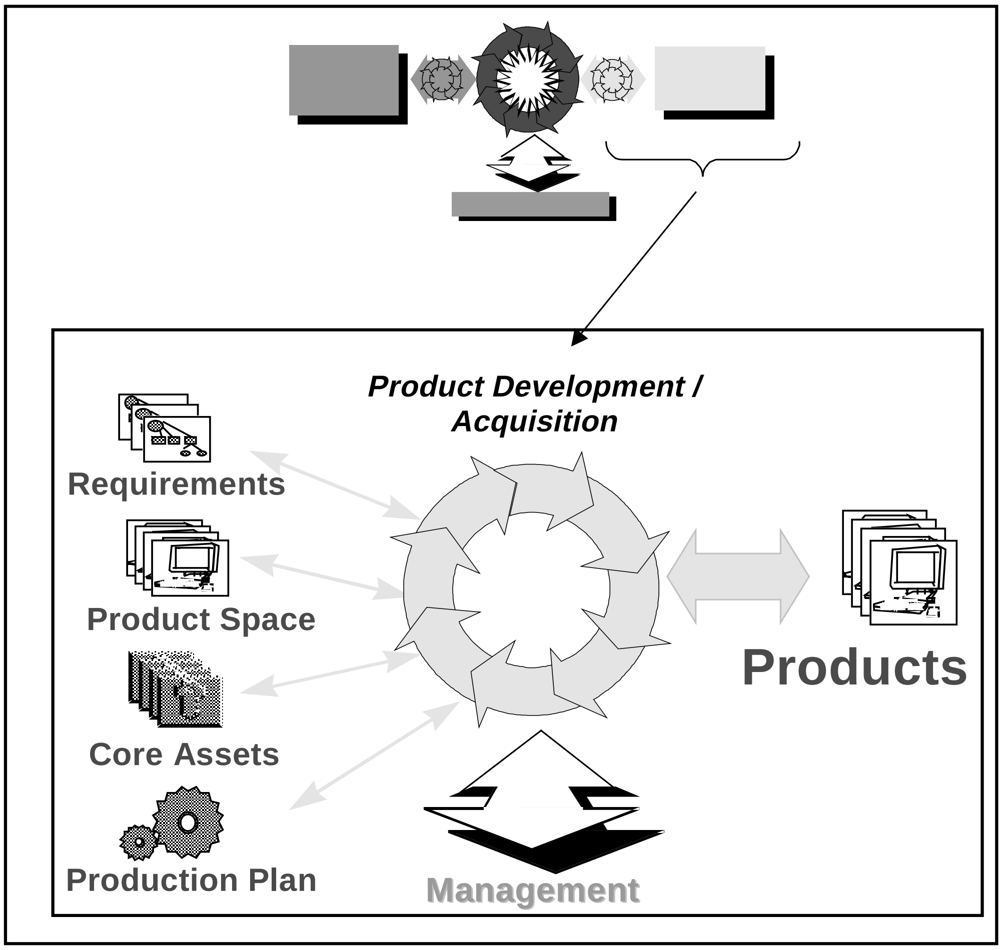


Figure 4: Product Developments/Acquisition

A product line is, fundamentally, a set of related products. Each product is produced by a group we call the *product group*. There may be one product group for several products or one product group per product. If separate, they may fall under the same organizational structure or be distributed widely across an enterprise. Sometimes a product line is commissioned—that is, one organization pays another organization to develop the core assets and one or more resulting products. The commissioning organization may or may not develop products itself once the core assets are in place. If it does, and the core asset organization also develops products, then the product groups may be distributed across entirely separate enterprises.

The creation of products may have a strong effect on the product space, core assets, production plan, and even the requirements for specific products. The ability to quickly turn out a particular member of the product line, perhaps one that was not originally envisioned by the people responsible for defining the scope, will affect the product space. Each new product may have similarities with other products that can be exploited by creating new core assets. As more products enter the field, efficiencies of production may dictate new system generation procedures, causing the production plan to be updated. And, in one of the most telling effects of a product line on its client base, a customer may change his requirements to bring them in line with the product line scope to take advantage of the quick time-to-market, reliability, and lower cost available with products in the product line.

Inputs for product production include the following:

- the requirements for any particular product, often expressed as a delta or variation from some generic product description contained in the product space. (Such a generic description is itself a core asset.)
- the core assets
- the production plan

Relevant practice areas include the following:

- requirements elicitation, analysis, and management
- architecture evaluation
- developing and implementing an acquisition strategy
- configuration management
- data collection, metrics, and tracking
- software system integration
- technical risk management
- testing
- tool support

Management

Management plays a critical role in the successful fielding of a product line. Activities must be provided with resources, coordinated, and supervised. Organizational management must set in place the right organizational structure that makes sense for the enterprise, and must make sure that the organizational units receive the proper resources (e.g., well-trained personnel) in sufficient amounts. Organizational management is the authority that is responsible for the ultimate success or failure of the product line effort.

Organizational management also contributes to the core asset list, by making available for reuse those management artifacts (especially schedules and budgets) for producing products in the product line.

If the product line is being commissioned, then the acquiring organization must also have an organizational management function that holds the responsibility for the success or failure of the product line acquisition effort. This management function must bear the responsibility for meeting or failing to meet the enterprise goals of the product line itself. The responsibility of organizational management in an acquiring organization is to oversee the contractor (the developing organization), and to ensure that the delivered assets and products are suitable and of high quality. If the acquiring organization actually takes over the core assets from the developing organization, and/or it produces products from those assets, then the acquiring organization also must have an architecture, core assets, and product groups.

Another kind of management that is required for product lines is the management of the organization’s external interfaces. Product lines tend to engender different relationships with an organization’s customers and suppliers, and these new relationships must be introduced, nurtured, and strengthened.

One of the most important things that management must do is create an adoption plan that describes the desired state of the organization (i.e., routinely producing products in the product lines), and a strategy for achieving that state. The adoption plan should lay out phase-in activities for various organizational units, prescribe training and skill development that will be necessary, launch a measurement program to gauge progress and success, and describe the eventual organizational structure that will be adopted.

Finally, management must either act as or find and empower a product line champion. This person is a strong, visionary leader who keeps the organization squarely pointed toward the product line goals, especially when the going gets rough in the transition stages. Every successful product line we have observed has an identifiable champion; most (but not all) failed product lines that we have seen lacked one.

In the area of management, relevant practice areas include the following:

- planning and tracking

- achieving the right organizational structure

- funding
- building and communicating a business case
- training
- organizational risk management
- operations
- developing and implementing an acquisition strategy
- customer and supplier interface management
- launching and institutionalizing a product line
- technology forecasting

References

[Bass 98a] Bass, L.; Clements, P.; Kazman, R.; Software Architecture in Practice. Reading, Massachusetts.: Addison-Wesley Longman, Inc., 1998.

About the Authors

Paul Clements is a senior member of the technical staff at the Software Engineering Institute. A graduate of the University of North Carolina and the University of Texas, he is a project leader in the SEI's Product Line Systems Program. His work includes collaborating with organizations that are launching product line efforts. He is a co-creator of the Software Architecture Analysis Method (SAAM), which allows organizations to evaluate architectures for fitness of purpose. He and others are working on an extension to SAAM, which will allow analysis of quality attribute trade-offs at the architectural level. He is co-author of *Software Architecture in Practice* (Addison-Wesley-Longman, 1998) and more than three dozen papers and articles about software engineering.

Linda Northrop has more than 25 years experience in the software development field as practitioner, manager, consultant, and educator. Her primary interests and contributions in the last decade have been in product line engineering, software architecture, object technology, and software engineering education. She has been with the SEI for the past five years. Prior to assuming the management of the Product Line Systems Program, she was lead for the Software Architecture Project, manager of the Education Process Project, and Chair of the SEI Education and Training Review Board. Linda is co-developer of the SEI Improvement Planning Workshop, and has taught software engineering at Carnegie Mellon University.

Before joining the SEI, she was associated with both the United States Air Force Academy and the State University of New York as professor of computer science, and with both the Eastman Kodak Company and IBM as software engineer. As a private consultant, Linda also worked for an assortment of companies on software development projects, assessed and recommended software process. She has developed and delivered a wide variety of software engineering and object-oriented training programs and seminars.

Spotlight 1

Software Product Lines: A New Paradigm for the New Century

Paul Clements

Software developed as a product line promises to be a dominant development paradigm for the new century, one that the Department of Defense (DoD) can leverage when acquiring software-intensive systems. This article discusses the advantages of product lines, uncovers some of their pitfalls, and shows by example the kinds of successes that the organizations can enjoy.

Imagine turning out a 1.5 million-line Ada command and control system for a Navy frigate warship. The system is hard real-time, fault-tolerant, and highly distributed, running on 70 separate processors on 30 different local area network nodes scattered all over the ship. It must interface with radar and other sensors, missile and torpedo launchers, and other complicated devices. The human–computer interface is complex and highly demanding. In this application, quality is everything: The system must be robust, be reliable, and avoid a host of performance, distribution, communication, and other errors.

Now suppose that you have not one of these systems to build but several. Your marketing department has succeeded beyond your wildest dreams. Navies from all over the world have ordered your command and control system. Now, your software must run on almost a dozen different ship classes including a submarine, and the systems are drastically separate in numerous ways:

- The end users speak different languages, and therefore the human–computer interface requirements are extremely different.
- The ships are laid out differently, have different numbers of processors and nodes, and different fault tolerance requirements.
- The ships employ different weapons systems and sensors.
- The ships utilize different computers and operating systems.

For all their differences, however, quality remains crucial in every system.

Suppose you are the manager for this megaproject. Do you panic? Do you resign? Run to a third-world country? What if you could produce each one of the systems for a fraction of the cost and in a fraction of the time that one would normally expect? And what if you could do it so that quality was improved and reliability and customer satisfaction increased with each new system? What if creating a new ship system were merely a matter of combining large, easily tailorable components under the auspices of a software architecture that was generic across the entire domain (in this case, of shipboard command and control systems)?

Is this a fantasy? No, it is not. It is the story of CelsiusTech Systems AB, a long-time European defense contractor. In the 1980s, CelsiusTech was confronted with the dilemma outlined above. It had to build two large command and control systems, each larger than anything that the company had attempted before, and it had barely enough resources to build one. Because necessity stimulates invention (and determination implements it), CelsiusTech realized that its only hope was to build both systems at once using the same assets and resources. And in a visionary stroke, CelsiusTech knew that its future lay in exploiting these assets for not only the first two systems but also for a whole family of products that the company hoped and expected would follow.

Software Product Lines

In short, CelsiusTech launched a software product line. A product line is a set of products that together correspond to a particular market segment or fulfill a particular mission. Product lines promise to become the dominating production-software paradigm of the new century. Product flexibility is the new anthem of the marketplace, and product lines fulfill the promise of tailor-made systems built specifically for the needs of particular customers or customer groups. What makes product lines succeed from the vendor's (and developer's) point of view is that the commonalities shared by the products can be exploited to achieve economies of production.

Product lines are nothing new in manufacturing. Boeing builds one, so does Ford, IBM, and even McDonald's. Each of these companies exploits commonality in different ways. Boeing, for example, developed the 757 and 767 transports in tandem, and the parts lists of these two decidedly different aircraft overlap by about 60 percent.

But software product lines based on interproduct commonality are a relatively new concept, and the community is discovering that this path to success contains more than its share of pitfalls.

The Software Engineering Institute has a technical program to identify and promulgate the best practices for product line production and help organizations negotiate the hurdles of adopting a product line approach. The Product Line Systems Program focuses on the following essential technology areas for product line production:

- **domain engineering;** reveals the commonalities and variations among a set of products
- **architecture**; the foundation for a product line, it provides the framework into which tailorable components plug
- **architecture-based development**; the disciplined derivation or generation of product components (and once the components are ready, whole products) from the architectural skeleton
- **reengineering**; helps mine reusable assets from legacy assets

The result is a technology infrastructure that can produce large custom systems quickly and reliably by checking out components from the asset repository, tailoring those components for their particular application (CelsiusTech uses compile-time parameters to instantiate different versions of a component), and beginning the integrate-and-test cycle as in normal system development.

Product Line Benefits

Once the product line repository is established, consider what is saved each time a product is ordered:

- **Requirements.** Most of the requirements are common with earlier systems and therefore can be used. Requirements analysis is saved; feasibility is assured.
- **Architectural design.** An architecture for a software system represents a large investment in time from the organization's most talented engineers. The quality goals for a system—performance, reliability, modifiability, etc.—are largely allowed or precluded once the architecture is in place. If the architecture is wrong, the system cannot be saved. For a new product, however, this most important design step is already done and need not be repeated.
- **Components.** The detailed (internal) designs for the architectural components are reused from system to system, as is the documentation of those designs. Data structures and algorithms are saved and need not be reinvented.

- **Modeling and analysis.** CelsiusTech reports that the real-time distributed headache associated with the kinds of systems that it builds (real-time distributed) has all but vanished. When the company fields a new product in its product line, it has extremely high confidence that the timing problems have been worked out, and the challenges associated with distributed computing—synchronization, network loading, and absence of deadlock—have been eliminated.

- **Testing.** Test plans, test processes, test cases, test data, test harnesses, and the communication paths required to report and fix problems are already available.

- **Planning.** Budgets and schedules can be reused from previous projects, and they are much more reliable.

- **Processes.** Configuration control boards, configuration management tools and procedures, management processes, and the overall software development process are in place, have been used before, and are robust, reliable, and responsive to the organization's special needs.

- **People.** Because of the commonality of the applications, personnel can be fluidly transferred among projects as required. Their expertise is applicable across the entire line.

Product lines enhance quality. Each new system takes advantage of all of the defect elimination in its forebearers; both developer and customer confidence rise with each new instantiation. The more complicated the system, the higher the payoff for solving the vexing performance, distribution, reliability, and other engineering issues only once for the entire family.

Clearly, product lines benefit the developing organization, but they also benefit acquirers of systems as well. Acquiring a family of related systems using a product line acquisition approach (as opposed to acquiring each system separately and independently) clearly falls within the realm of Department of Defense (DoD) reuse initiatives and policies, and it promises to accrue significant benefits for the DoD, including

- streamlined acquisition processes

- higher product quality

- lower acquisition costs

- simplified training

- reduced maintenance costs

Organizational Maturity Needs

It takes a certain maturity in the developing organization to successfully field a product line. Technology is not the only barrier to successful product line adoption. Experiences observed by the Product Line Systems Program show that it is equally vital to master organization, process, and business issues.

For instance, traditional organizational structures that have one business unit per product are generally not appropriate for product lines. Who will build and maintain the core reusable assets—the architecture, the reusable components, and so forth? If these assets are under the control of a business unit associated with one product or one large customer, the assets may evolve to serve that business unit, that product, and that customer to the exclusion of the others. On the other hand, to establish a separate business unit to work on the core assets but be divorced from working on individual products carries the danger that this unit will produce assets that emphasize beauty and elegance over practicality and utility. In either case, producing and managing the reusable assets means establishing processes to make the assets satisfy the needs of all of the business units that use them. This is a crucial role that requires staff skilled in abstraction, design, negotiation, and creative problem solving. The question of funding the core asset development is crucial.

Customer Management

Customer management becomes an important product line function. Customers interact with a product line organization in a different way. Marketers can no longer agree to anything customers want but must instead nudge customers to set their requirements so that they can be fulfilled by a version of the product line within the planned scope of variation.

Contrary to intuition, this often makes the customer much happier than before. Under the new paradigm, the marketer can point to specific requirements that would put the customer’s new system outside the scope of the product line, which would increase the cost and delivery time, lower the system’s reliability, and keep that customer out of a community of customers to which the vendor pays a lot of attention. Thus, the customer could clearly (and probably for the first time) see the real cost of those "special" requirements and make an informed decision about their real value. If the customer decides that a variant of the “standard” or product line system will suffice, so much the better. If not, the customer can still order a system to satisfy particular requirements but with a better idea of where the risks may be hiding.

The customer community should not be underestimated. In CelsiusTech’s case, the naval customers around the world banded together to form a users’ group. They did this in their self-interest—to provide a forum in which they could jointly derive new requirements for their evolving systems and drive CelsiusTech to supply new systems more economically than it otherwise might. But it does not take much to realize how beneficial this is to CelsiusTech as well: Its customer base is jointly defining the next generation of products and is effectively buying in to CelsiusTech’s approach, thus guaranteeing the vitality of the product line for years to come.

The users’ group provides a clear lesson for DoD acquisitions: It pays to collaborate (or at least communicate) when it comes to commissioning or purchasing similar systems.

Conclusion

The successful transition to product line technology requires a careful blend of technology, process, organization, and business factors improvement. The Product Line Systems Program is attempting to codify these practices and understand how they vary with the type of organization involved and the kind of systems being built. Through a series of workshops, case studies, and collaborative engagements, the SEI is helping to build a community of organizations interested in moving to a product line approach for their software products.

We believe that product lines will be the predominant software paradigm at the beginning of the new century. The history of programming can be viewed as an upward spiral in which the abstractions manifested by components have grown larger and more application meaningful, with resulting increases in the reuse and applicability of those components. From subroutines in the 1960s to modules in the 1970s to objects in the 1980s to component-based systems in the 1990s, software product lines will perpetuate the upward spiral by accomplishing previously unheard-of levels of reuse from system to system.

If the pitfalls are successfully negotiated, the result is an enviable capacity to deliver extremely large systems on time and within budget.

For more information about the Product Line Systems Program and its technology initiatives, as well as other product line information, see

http://www.sei.cmu.edu/programs/pls/pl_program.html

You can download the full report that details the CelsiusTech product line case study, which includes data about the company’s dramatic results in time-to-market, levels of reuse, and required staffing, at

http://www.sei.cmu.edu/publications/documents/96.reports/96tr016/96tr016chap01.htm

About the Author

Paul Clements is a senior member of the technical staff at the Software Engineering Institute. A graduate of the University of North Carolina and the University of Texas, he is a project leader in the SEI’s Product Line Systems Program. His work includes collaborating with organizations that are launching product line efforts. He is a co-creator of the Software Architecture Analysis Method (SAAM), which allows organizations to evaluate architectures for fitness of purpose. He and others are working on an extension to SAAM, which will allow analysis of quality attribute tradeoffs at the architectural level. He is co-author of *Software Architecture in Practice* (Addison-Wesley-Longman, 1998) and more than three dozen papers and articles about software engineering.

Spotlight 2

Product Lines in Practice at Three Major Corporations

Bill Thomas

Cummins Engine, Raytheon, and Hewlett-Packard all knew that they had a lot to gain from product line practice, and they were right: The companies have enjoyed substantial reductions in time-to-market, cost, and risk, and significant gains in efficiency and quality. But significant technical and cultural changes are also required, and all three companies have learned some valuable lessons from their product line programs.

Cummins Slashes Development Time, Costs

Cummins Engine Company launched a product line program for software in 1994, beginning with one legacy system for a diesel engine and modifying the software to extend it over several other engines.

Using product line practices presented difficult challenges on several fronts, says Joseph Gahimer, director of the Core Controls Group, which develops all core assets for embedded controls software and hands them off to application teams that apply the core assets to specific products.

First there is the complexity of Cummins's mix of engines, which range from light-duty engines (less than 200 horsepower) to high-horsepower engines (6,000 horsepower) used in a variety of applications, such as pickup trucks, large trucks, mining, rail, construction, power generation, and the military. Cummins, which has annual sales of more than $6 billion, sells to a variety of original-equipment manufacturers who all require varying degrees of customization to meet particular specifications. In addition, Cummins engines must conform to varying emissions regulations throughout its worldwide markets.

Prior to 1994, one team would develop each engine, including all the software for that engine. At that time, Cummins managers looked at the company's projections for new engine products and new features and calculated that it would need far too many engineers to develop enough new products to satisfy the expected demand. "The company realized that it would need to do something to get a lot of reuse" from one product to the next, Gahimer says.

Cummins wanted not only to develop many new engines; it also wanted to improve its time to market with those engines. A common reuse base would help that, too. Finally, customers were asking for more uniformity in the look and feel of Cummins’s engines, while still demanding customization for different products.

Five years later, Cummins has declared its original product line program a huge success. Before using product lines, it took Cummins 150-250 person-months of effort to develop software for the electronic control module (ECM) just to run an engine.

“It’s an embedded system, and we had to reinvent everything. Using core software assets, a lot of building blocks are available, and it’s a matter of assembling the building blocks and modifying them.” Cummins can now develop new ECM software to run an engine in less than 10 person-months. “In one instance, where we were applying the same fuel system across engines, we were able to build the software to run the new engine in just three days,” Gahimer says.

“Seventy-five percent of our product software originated from the core assets,” he adds. “That’s an incredible amount of reuse and it has saved a lot of cost across the company. Also, the reliability and quality are very good.”

Now that product line practices have proven so successful for software, Cummins is embarking on a second-generation effort, and is extending the product line concept to other areas of the company. “The company has seen the benefit and is saying, ‘Hey, where else can we apply this?’” Gahimer says.

Government Project Leads to Commercial System at Raytheon

Managers in Raytheon Company’s Space Systems Division examined the company's satellite command and control systems and saw that there was a high degree of overlap in software capabilities. Although the company did not have a product lines program in place at the time, Raytheon believed it could significantly reduce cycle time, cost, and risk by developing software only once and using it across its government and commercial markets.

Raytheon’s first product line effort, begun in August 1997, was a joint initiative with a U.S. government agency and was named the Control Channel Toolkit (CCT) program. The goal was to develop a reference architecture, extendible components, and reuse documentation that would cover most software requirements for satellite command and control on government satellite ground-system programs. In fall 1997, the CCT program brought in technical staff from the Software Engineering Institute’s Product Line Practice Initiative to serve as consultants on the program.

The effort was proven successful on the first reuse of the CCT product. Compared with its original bid, which called for largely creating the new system from traditional "code cloning" reuse, Raytheon reduced by 80 percent the lines of code engineers had to write. "That was a reaffirmation that by coming up with a common core set of capabilities—if we do the domain engineering right—a large majority of the capability should be there," says Jeffrey Shaw, who served as program manager for CCT.

Shaw says thorough domain analysis is critical—not just for identifying common capabilities but also the hotspots where things tend to always vary among spacecraft and missions. Knowing about the variability helps Raytheon engineers develop well-engineered "variation points" that can be easily adapted to new applications, which helps to further reduce costs and cycle time.

Shaw points out that in reusing CCT, Raytheon engineers spend significantly more time on software architecture and design work, but afterward the implementation work proceeds relatively quickly. Based on that experience, Raytheon is beginning to adjust its cost and bidding models.

From the beginning, Raytheon planned to extend the experience it gained with CCT by applying that experience later to commercial products.

Shaw has since moved on from the CCT project to become product line manager of Eclipse, which, as Raytheon intended, will be a commercial application of the principles learned with CCT. The existing Eclipse software architecture, which is object-oriented and modular, will be migrated to a true product line architecture.

Unlike CCT, which is a toolkit with a reference architecture, Eclipse is an end-to-end system. Raytheon's transition plans call for moving components from the existing system, one at a time, into a product line architecture. "One key in any product line plan is to get the common baseline under control, and to then have a well-planned strategy for migration."

Evolving Platforms at Hewlett-Packard

Hewlett-Packard Company has long developed hardware products on common platforms, but platform development for software is relatively new—within the past 10 years for many applications, says Emil Jandourek, the Practice Area Section Manager within HP's Product Generation Solutions (PGS) group. PGS partners with HP's product divisions to help them evolve their product-generation capabilities and competencies.

The company’s product line approach for software involves whole products, such as families of printers, and tends to be very effective when it drives products that have multiple simultaneous releases and involves rapid time-to-market schedules.

As of 1997, core assets made up 89 percent of the software in one line of HP printers. The other 11 percent consisted mostly of adaptations that matched the software to a particular printer in the line.

A common Hewlett-Packard strategy is to evolve new features into an existing platform. For example, duplexing (printing on both sides) was originally enabled on only one printer, but is now available to all printers, though not all printers use it because they lack the necessary hardware.

“We’re doing things on the hardware side, coupled with the software,” Jandourek says. “An overall platform approach is emerging in both disciplines: one that leverages what you have and extends it forward. In multiple simultaneous releases, you can reduce effort between products and have a shorter time to market.”

The next step in HP’s product line approach involves “an overall shift from a more monolithic, integrated, and tightly coupled asset base to a modular component-based architecture with corresponding assets,” Jandourek says. This component-based approach is more efficient and gives developers more flexibility. The challenge is to correctly establish the boundaries for the components.

No Universal Prescription

Jandourek points out that the product line approach “is not a universal prescription.” For example, there would be too many unknowns with a startup product for a new customer base. It would be a poor strategy to delay that product in order to develop a product line platform because the product could fail—and perhaps take the entire organization down with it. “But when it’s a product category that is well established, even if it’s new, you could design it to be platform-based.”

For Cummins, the greatest challenge has been to conduct good domain analysis—a process that only comes with experience. “With product-line engineering, until you do it for the first time, you don’t have a feel for what it takes to do it,” Gahimer says. “Our domain experts conducted domain analysis based on their experience, but we identified many improvements later. So the lesson is that you need thorough domain analysis to help you anticipate future needs. Still, you can’t forecast or project all the different uses for the product line until you go through it.”

Cummins also learned that it is important to get the workflows and rules established, and the necessary tools and training in place for the product line effort, which Gahimer says make up the infrastructure that keeps the effort moving over the long term.

The product line effort also requires discipline. “There are a lot of conflicting pressures during development,” Gahimer says. There could be a desire to quickly get software written for a specific application, and some members of the organization might not want to take the extra time to make that software available for common use. But unless the software is available for common use, the next group will also have to start from scratch.

Finally, Gahimer says, the architecture for software product lines has to be portable and able to move between different products and technologies because the lead time to create a new architecture is often longer than the company’s ability to predict the market's needs.

Raytheon’s Shaw also emphasizes the importance of the product line’s software architecture. “The whole key is architecture,” Shaw says. “We’ve gone through object modeling, object orientation, component-based reuse—but those were baby steps. It’s not until you have a reusable architecture that provides a context for all those enabling technologies that you can reap the benefits that everyone’s been trying to get for 20 years.” With that architecture now in place, Shaw says, “we’re excited. We’re seeing the bottom-line business numbers.”

Management Champions and Cultural Change

Gahimer also points out that Cummins’s program might have failed without a champion in senior management, which for Cummins was the vice president of electronics. “He had the vision, the authority, and the persistence to establish the first generation concept and keep it flowing.”

Raytheon has also learned that the organization and its culture have to change to support product line practices. Shaw explains that with Eclipse, Raytheon is marrying two architectures: a high-end system architecture and an underlying product architecture. On one hand, the product line organization is looking for core assets to be used across products, but on the other hand, individual programs are used to developing “stove-piped” systems, with a single end use in mind. “The biggest question is: How do we integrate processes and have cultural change to get the overall efficiencies that we need from product line practice?” Shaw says.

Such efforts can expose underlying concerns, even at the best companies. “Stove pipes typically involve someone’s sphere of control,” Shaw says. “Despite everyone’s desire to be good company people, there is a lot vested in the current organization.” Shaw says the solution is to have alignment at the senior-management level. “It flows down from there. Everyone has to see that there are incentives to adopting the new culture. If they think they will end up as losers, they’ll fight it.”

Roundtable

A Scenario for Using the Product Line Practice Framework

The following discussion was part of a panel presentation on the SEI Product Line Practice Framework, Version 2 (please see this issue's Background article) at the 1999 Software Engineering Symposium, Aug. 30-Sept. 2, in Pittsburgh. The participants, all senior members of the SEI technical staff in the Product Line Systems Program, were:

- Linda Northrop, Product Line Systems Program director
- Larry Jones,
- Sholom Cohen
- Dennis Smith
- Paul Clements

The topics discussed in this Roundtable include:

- launching a product line
- scoping a product line
- the use of commercial off-the-shelf (COTS) software in a product line
- product line architecture

Northrop. The SEI's Product Line Practice Framework is a static body of knowledge. It is impossible to provide a univeral prescription as to how all organizations should apply this knowledge, since each organization's context and starting point are different. So we decided to help you get an idea of how you might use the framework in your organization, and give you a dynamic -- and dramatic -- look at the framework. We're actually going to provide you with a scenario that suggests how to use the framework in your product line activities. In order to add this dynamism, we're going to do some interactive discussion. I won't go so far as to say that this is drama. There's a real reason why we have day jobs. Our dramatic skills are not nearly as good as our technical skills, I might add.

My name is Ms. P.L. Wannabe and I'm a program manager. I have business goals; I know what my business needs to do, but I'm having difficulty addressing those goals with the technologies and processes that we employ. Now, I've seen this SEI Product Line Practice Framework that looks promising, but I'm clueless as to how to use it. I consulted my favorite book on how to manage and I still didn't come up with the answers on how I'm actually supposed to solve this product line problem. So, I decided to hire some consultants. These consultants are going to help me sort through this product line practice framework and see how I might be able to apply it. In a word, I want to start using software product lines. I have four experts with me who will talk about launching a product line, scoping, architecture, and COTS [commercial off-the-shelf software].

My company is a consumer electronics company. Basically, I have some business goals. I want to improve my time to market and I want to increase productivity. We have an excellent reputation. We are world renowned for the quality of our products and we have greatly satisfied customers. I don't want to do anything to mess with a good situation to improve time to market. But if I don't improve time to market, I'm going to be out of business. My problem is that I've got a lot of software in my products and I never used to have a lot of software in my products. And software in my products seems to be more complex. These software costs are killing me. So, you know, I thought about adding more people, but the fact is if I add more of these software people they're expensive, it eats my profits. And secondly, there aren't enough of them out there. So, even if I -did- have the money to hire them, I can't seem to be able to hire them. I've currently got about 250 people in my organization and that's what I'm going to have to work with. I've been thinking about what we could do with reuse because it seems we build a lot of similar systems. We've tried libraries of code, but they just didn't work. We didn't get any significant reuse and we didn't get any payoff from the reuse we did get. So, I read this SEI stuff about product lines and I got this framework document, but I don't know how to sort through it. So, I'm asking you if you would help me start a product line.

Cohen. I think my colleagues would agree with me, we need to know more about the company and the kind of products you produce. So give us some more background information about the company.

Northrop. I'd be glad to do that. We build audio systems -- all kinds of audio systems -- personal, home, automotive.

Jones. How are you structured in order to build these products?

Northrop. My organization is divided into projects and each project builds a product.

Clements. You're suspecting that there's commonality among these products. Is there currently any reuse?

Northrop: I know there's commonality. Actually, any reuse is opportunistic. It's up to the project managers and if they want to do reuse they do it; if they don't, I guess they don't. And mostly they don't.

Smith. Have you heard of the software CMM?

Northrop. Who hasn't?

Smith. Have you been assessed or have you assessed yourself?

Northrop. Well, we actually haven't been officially assessed, but we did a self-assessment and I think we're about Level 2.

Smith. Let me test my understanding of what you've been telling us. Your company builds consumer electronics. You have a great reputation and you've got about 250 people who work for you in the software area. You've read what the SEI and others say about product lines and you see opportunities for achieving economies of scale and increasing quality through product lines. You have three main audio products, for the home, for cars, and personal products. Your projects now are all organized around individual products so I imagine that very little sharing of information goes on between those products.

Northrop. Yes, probably not.

Smith. Your reuse is opportunistic, at the discretion of the individual project manager. And you have been self-assessed at CMM Level 2.

Northrop. You have the picture. Now, how do I start a new product line?

Jones. First of all, you need to understand that launching a product line effort requires many of the practice areas that you're going to find throughout the framework.

Software Engineering Practice Areas:

- Domain Analysis
- Mining Existing Assets
- Architecture Exploration and Definition
- Architecture Evaluation
- Component Development
- Testing
- Requirements Elicitation, Analysis, and Tracking
- COTS Utilization
- Software System Integration

Technical Management Practice Areas

- Data Collection, Metrics and Tracking
- Product Line Scoping
- Configuration Management
- Process Modeling and Implementation
- Planning and Tracking
- Make, Buy, Mine, Outsource Analysis
- Technical Risk Management
- Tool Support

Organizational Management Practice Areas

- Achieving the Right Organizational Structure
- Building and Communicating a Business Case
- Funding
- Market Analysis
- Developing and Implementing an Acquisition Strategy
- Operations
- Training
- Customer and Supplier Interface Management
- Technology Forecasting
- Launching and Institutionalizing a Product Line
- Organizational Risk Management

Northrop. I have to use all of those practice areas?

Jones. Well, many of them. But the depth and degree to which you have to go into each of these is going to depend on the scope of your effort.

Northrop. What exactly does that mean?

Jones: Well, you ought to really try to treat launching your product line as a special type of technology change effort. You'll see that you're not alone in the approach to this. There's been a lot of work that's been done on technology change. Our colleagues in the adjoining sessions working on process improvement have done this for years. So I want to encourage you to, first of all, draw upon this existing body of knowledge. Among this is the fact that you need to account for the human aspects of change. It's not all going to be technological. You've got to treat technology change, itself, as a project. Lastly, I'd like to suggest that you follow some sort of iterative technology change model.

Northrop. Could you suggest such a model?

Jones: Yes, there has been some pretty good work done by the Software Engineering Institute in Pittsburgh. They've come up with a model that has had a great deal of success in process improvement. It's called the IDEAL model. With a little bit of artistic license, it's very easily adaptable to any type of change project. So let me very briefly run you through the kinds of steps that are involved. First of all, recognize that it is iterative and that you will do each of these steps -- perhaps to some degree -- more robustly than others, depending on where you've started on this technology change. The Initiating phase of this model requires you and your organization to recognize that you have the need to change and to establish appropriate infrastructures to deal with this and allocate resources. In Diagnosing, we take a look at where we stand at a particular point in time and what we might be able to do by way of possible improvement. In the Establishing phase, we plan, based on the priorities determined in our diagnosis. In the Acting phase, we execute those plans. Lastly, we want to learn lessons in the Learning phase, and apply them to future IDEAL cycles.

Northrop. This all sounds wonderful, but how am I going to do this in my organization?

Jones: Let me try to give you some specifics. First of all, I suggest that you try a limited cycle of the IDEAL model, and devote it to concept exploration.

Northrop. How would that go?

Jones: This should be an inexpensive try at this. First of all, you want to commit some limited resources to explore the applicability of the product line approach. You might designate a person to go out there and gather data and run this concept exploration. Next, a diagnosis would be somewhat simple. You might do a sort of back-of-theenvelope sketch of what product line technologies are available and how that might relate to your organization in particular. In our establishing phase, we’re going to take the interesting case, which is to assume that you’ve made a “go” decision, and plan for the next round since we’ll assume you’ve decided that further exploration is warranted. Typically at this stage, the acting and learning phases are truncated or are rolled into the next IDEAL cycle.

Northrop. So, what do I do after this Exploration stage? Clearly this doesn’t launch the product line.

Jones: No. But we could try another round with IDEAL, which we might call concept refinement and initial implementation. In this case, we start to beef up the steps that go in each phase. In the Initiating phase there would be more awareness-building and advocacy-building. This should be broader than just the few people who were involved during the Concept Exploration cycle. You would have to commit more resources and you might want to establish more structure in order to tackle this. Your Diagnosis would be more elaborate too, in which you review your existing organizational environment, your assets and culture and the things that you might want to take stock of. In your Establishing phase, you’re going to start off with putting your toes in the water, and developing visions, goals, strategies, and objectives for the product line. You want to take a look at some of the commonalities, scope the product line, and develop the preliminary operating concept, in which you describe how this is going to play out. This can be a significantly useful activity for risk mitigation. You will also then worry about preliminary architecture definitions, preliminary mining of your existing assets, which you say are out there, and you will work on developing a business case that’s going to establish why you would do this. Our Acting phase is then to execute that plan. And then in Learning you will select lessons learned from what you’ve done in that first cycle, refine your approach, and take another look at whether you proceed.

Northrop. How do I make sense of this Learning phase? How do I go ahead with it after that?

Jones. Let’s say we’re going to proceed with this, that a product line approach has some viability in your organizational context. I won’t belabor yet another cycle of IDEAL, but you can consider that to be superimposed over this. You might beef up some structures and possibly do some rediagnosis, based on the first cycle of learning. But we want to refine our approach to create what I call a product line adoption or implementation plan.

Northrop. What might be in a product line adoption or implementation plan?

Jones. You’re going to have refinement of your initial architecture definition and you’ve learned some lessons about that, based on the first cycle. You want to have some further mining of your legacy assets. You can refine your concept of operations and get more details on how you'll actually tackle that. The details of how to migrate then from the current organizational state to the future state is what's really going to be described in this concept of operations.

Northrop. I just don't like the whole word "migration." It sounds like a tremendous amount of upheaval. Can't I use this product line approach without upheaval in my organization?

Jones. I'm afraid not. Execution of this plan is going to really rely upon you and your leadership skills. Let me give you some practical advice on how you might tackle some of these things that might otherwise be overwhelming. You're going to have to show leadership in establishing the goals, communicating, and being an advocate and cheerleader for these things. You're going to have to allocate more resources. You're going to have to set up proper organizational structures and modify the reward system to support the effort. You're going to have to pay a great deal of attention to those initiating phase activities, particularly if they've been somewhat short- circuited while you tried to feel your way around the first couple of cycles.

Northrop. Are you suggesting that I have to personally become actively involved?

Jones. Yes, you most certainly do.

Northrop. I'm overwhelmed.

Jones. Let me see if I can take some of the pressure off by giving you this whale sandwich a few bites at a time. You've already said that you're a pretty good planner because you're Level 2 CMM. So, if we actively manage this change as a project and don't just let it happen, you'll be a lot happier with it. Part of that is based on your excellent reputation in the hardware industry; you might be able to manage your risks pretty well. A way to keep the scope down, and to learn your lessons in a small way before making big mistakes, is to use pilot projects actively as part of your risk mitigation strategy. Initially, don't go too hard and fast on some of your processes and organizational structures. Some organizations find a great deal of success using what you might call "lightweight" processes and organizational structures to tackle this initially. Keep things flexible and less formal to learn what works in your organization before you put a great deal of turmoil on yourself and your organization.

Northrop. That makes me feel a little better. I'm actually thinking that I know how we can come up with a goal, a vision and a strategy because we have a total-quality environment and I have people who I can enlist to help me. But I'm really perplexed. One of the things you said to plan, which I didn't understand at all, is "commonality analysis and scoping." Early on, you also said that the whole impact of the effort is really going to be dictated by the scope of the product line. I have a problem. What's the scoop with this scope business?

Cohen. Let me help you learn a little bit about scoping and some things you might want to consider. You mentioned that you know what it means to establish a vision and set goals. With product lines, you have to go back and do the same thing: What's your vision for the product line? What do you want your company to achieve? What are the goals that you've set for yourself? So, if you've set the vision, you probably want to consider the things you want from the product line: What should it accomplish in business or technical terms? You mentioned up front that you need to reduce costs and control your growing workforce. These are pretty good goals that a company would want to achieve by taking on this product line work.

You also need to know what steps you want to follow to make those things happen. Again, with launching a product line, you need a strategy, a concept of operations. What steps are you going to take up front in order to make this play out and work successfully from day one? At that point you know where you're headed, you know what your vision is long-term, and you know what things to do to get started. At this point it's appropriate to start defining the product line, or possibly product lines.

Northrop. OK, I get this. Is this scoping?

Cohen. Well, scoping is actually about determining boundaries. What's inside the product line and what's outside. How do the products differ based on personal, car, or home use? You need to know who your end users are. What service is the product going to provide to users? Looking at some of the products, some have built-in radios, some have playback and also record.

Northrop. So essentially, I have to look at the variations.

Cohen. What you really want to look at in terms of the scope is: What issues or decisions will drive the architecture? You're going to make a product line architecture for some products and not others. So what are some of the issues that come up here? For example, something that a person could carry is not going to be as integrated as something that is used in the home. Those are things that are going to affect the architecture of the system. You have things that are going to be able to pick up a broadcast at home or in the car. Or you may be talking about an interface to pick up live broadcasts over the Web, or the capability to download Web source material and play that back. You may be able to download that off the Web and take it with you. What are the external entities to support that and what are the interfaces?

Another problem to consider during scoping is: Who is going to distribute these products? For the auto version, is it going to be the original equipment manufacturers -- GM, Ford, or Daimler-Benz? Will you work with after-sales organizations? Then you want to look at future plans. We've been looking under the microscope at these products, but you also need to take a longer vision, and see where in the future these products might go. The Web might be one thing that affects that.

Northrop. You brought up an excellent point. How wide is this scope?

Cohen. When you do product line scoping, you really want to look at scoping the products we’ve already mentioned. Then try to pin down exactly what the product line or lines are going to be. You might have one product line of entertainment products and develop an architecture that covers all the products in that product line. That accomplishes one goal. You might be able to use one kind of user interface across all of those. We call that “economies of scope” because one architecture encompasses a broad scope. The tuners and playback devices may be able to be reused. On the other hand, there’s probably a longer upfront development effort. You’re going to have to do scoping or architecture work and look at a broad range of products before you’ll be able to implement your first one.

On the other hand, you may want to look at a separate product line for each market, so that would mean different product lines for home entertainment systems, car systems, and personal products. There may be easier adoption, or earlier upfront kinds of implementation. This approach may lead to possible fragmentation, so you may lose some of the economies of scope we mentioned earlier. There could be other segmentations. You could develop a tuner product line or amplifier product line, and so on. You may want to consider outsourcing some of the development.

Northrop. All of those things are internal. Should I look outside my company?

Cohen. Yes, I suggest you consider some of the things that are going on out in the field so you don’t lose sight of what other people are doing. Market studies are appropriate to see what the competition is doing. Your competitors may be doing some things you want to look at. You may want to look at opportunities for shared development, collaborations, opportunities to combine Web-based products with things you’re doing right now. COTS [commercial off-the-shelf software] is another piece of information you may want to look at: how to incorporate off-the-shelf capabilities into your products. And then there are other interactions. How do you do upgrades? Is it possible to download new software for a system or for the vendor to send a new disk or CD? Media types may include digital CD, DVD, or other compact-disk formats. Those are the kind of considerations you want to make.

Northrop. Hmm. So after I do all this scoping, what’s the result?

Cohen. We hope that you’ve identified what product line you want to invest in, or product lines. You should have an understanding of the kinds of services that you’ll provide to users and what kind of product integration you’re going to be able to accomplish. And you really want to bound your variability. You want to be able to say, “These are the products we’re going to do and these are the variations that we might be able to accommodate.” You might want to say, in the future, that you know that products will be integrated with the Web. You may not be able to do that today. So you may want to leave that as an opportunity for future growth, and you want to set a plan for product line development, common features, variations, architecture drivers, and so on. The SEI has produced guidelines for developing a product line concept of operations that should help you get started. [Please see Guidelines for Developing a Product Line Concept of Operations.]

Northrop. This is beginning to take shape in my mind, but something you said back when you were talking about market studies is bothering me. You said something about COTS. Now I'm confused. Why and how would I want to use COTS in my product line?

Smith. You've already seen that product lines offer tremendous examples of economies of scale. For example, organizations such as Hewlett-Packard and Motorola have achieved economies of scale and they have shortened their time to market to give themselves an edge in today's fast paced world.

Let's now add COTS to the equation. COTS gives you a good news/bad news scenario. First for the good news. I can go down to a computer store and buy a COTS product that costs me $89.95. If your organization were to build the same product from scratch, it would probably take about two years and cost about $3 million. COTS lets you multiply the leverage that you gain by moving to a product line approach.

Northrop. While it makes sense, and dollar savings really appeal to me, isn't it kind of risky?

Smith. Absolutely. That's the bad-news part. You really don't know what's inside the box of a COTS product, and you certainly have no control over what's inside of it. You can see what the vendor says on the outside and there may or may not be a manual to expose what's in there and, especially, to let you know what the interfaces really do. So, it's a black box and you're certainly taking risks. In addition to that, you don't have control over the evolution of where the product is headed. Also, in general, when you're dealing with COTS products in a product line, you need to develop your own requirements very flexibly. Your architecture needs to be flexible so that you can handle a wide variety of interfaces.

Northrop. That's way too general. How exactly do I do this?

Smith. Well, if you want to use COTS successfully, first of all, you need to really understand your own architecture, especially the points of variability and flexibility. You need to consider the policies and procedures that your organization uses for acquiring software. So, for example, you may be able to go down and simply buy a product from a computer store, or it may take you six months to be able to do that, depending on your organization. You need to also have a set of documented approaches for scanning the marketplace. There may be dozens of competing products that have some potential to work as components for you. These products will most likely all be imperfect fits, and they may or may not get better. You need to have a very good way to evaluate these products and not just take the blurb that's on the back of the box, or what a vendor happens to be telling you. Once you acquire a product, you also need to perform very extensive integration testing.

Northrop. Are there more things I need to consider?

Smith. There sure are. If you want to really use COTS successfully in a product line, once you've qualified the potential components you're going to need to do a fair amount of adaptation. Despite what your favorite vendor may say, these will not simply plug and play into your product line. You may need to adapt them to work within your architecture through wrappers, software, middleware, or other glue code. In addition, once you've done that, you need to assemble these components. You need to account for interactions between the COTS components and other components of yours, including your middleware. In addition to that, the COTS people are going to make periodic updates. You may want to move to a later release or you may not. That's going to depend on the goals of your product line, as well as where the COTS products are headed. It's also going to depend on your assessment of this vendor. They may not be in business in five years, or their evolution strategy may be in direct conflict to your product line approach.

Northrop. I'm confused. This seems like a lot of information about COTS, but how does this all relate to my product line?

Smith. Specifically, a bunch of potential COTS products could become core assets in your product line. For example, COTS databases, graphical user interfaces, or middleware all represent potential candidates for core assets. I was recently working on a project on which we found a graphics component available on the Web that cost $50. We were integrating a set of legacy modeling components into the architecture. Most of these components had their own graphics packages, resulting in inconsistent and confusing representations of the outputs. By investing $50 -- and with quite a few adaptations, I might add -- we were able to have a common set of graphics that went across all of these components.

Other issues when dealing with core assets include things like stability and maturity of the particular product and vendor. You want to know how long a product has been around, and what its reliability is. You also want to understand how it interoperates with the other components that are in your architecture and how its interfaces and protocols compare to your own standards and protocols.

Northrop. So, what you're telling me is that I can use some of these COTS pieces as core assets, but I need to pick out my own architecture and dictate how it all fits together. Is that true?

Smith. Yes it is.

Northrop. But now I want to build products. How do I apply COTS when I'm building products?

Smith. COTS products could very well become components for your products. But, there is a whole set of questions that you need to examine. Certainly one issue is that of functionality. What does the COTS product actually do? Not only what does the vendor say it does but what does it actually do, how does this meet your needs, and how does it compare to other products in meeting your needs? There's the issue of looking at quality attributes, such as availability, performance, security, or modifiability. You need to be able to articulate the most important quality attributes for your system, and see how well these potential components enable you to meet these requirements.

Another issue is cost. There's certainly a strong cost avoidance up front, but you need to look at the whole set of lifecycle issues. A package may fit 80% of your actual needs, but how important are those other 20%? So, you may be giving up something with COTS also. When you look at a COTS product, you need to also consider the flexibility of fitting the product into your architecture or updating it. Many vendors will tell you that their product is plug and play, but that's not necessarily the case. You need to know specific mechanisms for how it interacts with your system.

Northrop. That's a big risk. Are there other risks?

Smith. There sure are. COTS components will have unknown interactions with other COTS products and with your software. So, you certainly need to worry about how well behaved that piece of software is. The updates of different COTS components are not synchronized -- they're going to be coming at very different times and they will be doing very different things. You need to consider that. When you replace a COTS product, you're not going to replace the components on a one-for-one basis because the components are built for very different purposes and they do very different things. You're not going to have exact matches when you do your updates. As we all know, configuration management is critical for product line approaches, and certainly when using COTS within product lines. As I said before, COTS products also need to be adapted to fit into your architecture.

Northrop. Everybody's talking about architecture. What is it and why is it so important?

Clements. Software architecture is, in my view, one of the most important—if not *the* most important—core assets that you're going to build when prosecuting a product line. We like to say that software architecture is the structure or structures of the system. These structures comprise software components, the external divisible properties of those components, and the relationships among them. Remember that the product line's scope defines the commonality that all the products will share and the variability that the different products will feature. Architecture goes hand-in-hand with that scope. It is the carrier of the commonality. We build so that all the products in the product line will share the same architecture, the variation mechanisms that we've built into the architecture. We build products by using the variability mechanisms of the standard architecture to achieve instance architectures for the individual products.

Architecture is the carrier of the quality attributes in a system. By "quality attributes" I mean things like performance, security, and modifiability. So, every product in the system must meet the basic quality requirements, but the scope may identify different products that have different quality requirements. You might have a high-security product, a low-security product, a high-performance product, a low-performance product, all within the same family. You need an architecture that's flexible enough to permit all of those things.

Architecture is the basis for flexible assignment of personnel among your projects. You have 250 people right now working on individual assignments, and their expertise is probably limited in large part to the product that they're working on right now. Wouldn't it be nice if you could swap those people back and forth flexibly? A common architecture will let you do that because when people learn and are fluent with the common architecture, then they're automatically fluent with the different products that use it. So architecture is, again, the embodiment of the product line's commonality. The right architecture is going to help you achieve success in this product line. The wrong architecture is a recipe for disaster.

Northrop. The thought of being able to leverage my people in a better way is really appealing. As I told you, I just can't hire any more people, so I've got to make use of my 250 people in a much more productive way. It occurs to me that I've got to have some really special people to do architecture. What kind of people and how many people do I pick to build this architecture?

Clements. You want to work inside of your organization if you can. We find that one of the essential qualities for product line success is long, deep, domain experience, and you probably have that in your organization. Bringing people in from the outside is probably not the thing to do. Ideally, you will have senior designers in your organization who have domain experience. You should choose from among them. And, ideally, you'd like to identify a single, gifted individual or a small, cohesive, integrated design team to turn out the architecture for the product line. We tend to be suspicious of architecture by committee as a modus operandi, though I guess there are some successful examples of that. Conceptual integrity is one of the most important qualities of the products in the system. You'd like that to flow from the mind of a single architect, or a small group of architects with a clear leader.

So, you should look for people in your organization who are gifted in this area. They need special skills. First of all, they have to be technically savvy. You want them to be able to understand the latest technology. This will let you survive and move into new markets as the technology changes. They have to have really good communication skills—both input and output. For input, they have to listen to competing requirements and desires on the part of all the product manufacturers. They have to understand what is needed and expected of the architecture. They have to gather all the inputs necessary to make that architectural decision. On the output side, they have to be able to clearly communicate the vision to people and to make them understand why the architecture is going to solve the problem at hand. Thirdly, the architects all have to have people skills. Architecture is the medium through which tradeoffs and conflicts are negotiated. One person is going to want really high security, but another won't want to pay for that because he or she doesn't need it and instead wants more performance. Those conflicting needs have to be negotiated, so the architect is often at the center of the whirlwind of controversies. The architects have to remain calm and communicative.

Northrop. I'm picturing an individual in my organization who could be the architect. I would need to give her more information about a product line architecture. So, given that she has the right set of skills, what extra would she have to know about a product line architecture to be able to do the job?

Clements. Well, product line architectures are not so unlike ordinary system architectures in that there are requirements that they must meet. They must allow their systems to meet behavioral and quality requirements. For a product line architecture, there's a plurality of products for which that must be true. So, each architecture must allow the systems to meet their behavioral requirements: They must allow you to play music, tell what time it is, get songs off the Web -- whatever functionality is required. The architecture must also let the systems meet their quality requirements: performance, security, and modifiability in particular. The architecture must meet the developing organization's ambitions. The model that we all learned in software engineering class -- where somebody writes down the requirements for a system and then tosses that document over the transom where it lands with a thud on the architect's desk, and that's where the architecture comes from -- that model's wrong; it's a lie. The developing organization has a lot of subtle influence on an architecture that will decide whether it's successful or not. You might have a group of underutilized people that you'd really like to put to work. The architect may need to take that into account. You may have a tool environment that you've already built. It would be a bad architecture that wasn't compatible with that tool environment. In particular, your ambition is to have a product line, so the architecture needs to be able to encompass the commonality and account for the variability across all the products to satisfy that organizational ambition. Finally and critically, and I think obviously, the architecture must be buildable. The most elegant architecture isn't much use if it takes 300 people and six years to turn out a product line.

Northrop. This may sound like a really naive question, but do architects inherently know how to do this?

Clements. What I think you're asking is this: "Do they have to do this anew each time?" The answer to that is almost certainly no. One of the reasons that architects are senior people is that they have experience over the years designing similar systems. The architecture of a product line is probably not going to be created from scratch; that would be highly risky. Architecture typically comes out of the architect's prior experience. We see architects who have had great experiences with certain design approaches, and those things can be used again. We also see architects who have had bad experiences with design approaches, and you're not going to see them repeat that for a while, even if they might be the right answer for the next system. The architect should also be familiar with other similar systems in the domain, which is one reason you would pick that person: her long, deep experience in the domain. She will bring that experience, and those observations to bear on solving this problem.

The community is now actually coming to the rescue with architectural patterns and architectural styles. The subject of attribute-based architectural styles is something the SEI is working on. These give architects the beginnings of a kitbag that they can bring to bear to solve a particular problem. The architecture is also going to be heavily influenced by whether or not we want to bring in components that we have mined from legacy systems.

In addition, there's one other factor that the architecture needs to have. We often talk about architecture in a descriptive fashion -- it describes the system. But architectures need to be prescriptive as well.

Northrop. What exactly do you mean by prescriptive?

Clements. The product line is only going to work if the people building the products use the architecture that's been laid out for them. In that sense, the architecture is prescriptive -- it's before the fact. It's not just what was done and how the products were built, it's the marching orders for the product organization. The people working on the product line have to be comfortable with architecture-based or architecture-driven development. They have to understand how to take the specifications and write code that conforms to it, because that's how commonality comes to be exploited.

Northrop. Documentation must be really important.

Clements. Documentation is critical, especially in a product line architecture. Architecture in any system is a vehicle for communication. It lets the stakeholders communicate with each other. That only happens in a disciplined manner if the architecture is well documented. In a single system, you could imagine that an architect could take his or her vision to the product builders and sit with them and coach them, talk to them, and make sure that all of their questions were answered. A lot of architects spend their time doing that. In a product line, there are going to be 8, 10, 12, 15 projects all using the architecture. It's just not viable for the architect to be like a butterfly pollenating all the flowers. So in this case, the documentation is especially crucial. It's going to have to be clear and unambiguous, and it's going to let the product builders know enough to be able to do their jobs.

An architecture is also the vehicle for analysis. A system has to meet its quality attributes, but in a product line all the systems have to meet all their quality attributes. So architecture becomes very important as the basis for analysis of a product line.

Northrop. I do have a lot to think about, but I certainly have a much better understanding than I had before.

About the Panelists

Linda Northrop has 30 years of experience in the software development field as a practitioner, manager, consultant, and educator. She is currently director of the Product Line Systems Program at the Software Engineering Institute (SEI). The Product Line Systems Program works in the areas of software architecture, reengineering, and product line engineering. Northrop is the former chair of the SEI Education and Training Review Board and co-developer of the SEI Improvement Planning Workshop, and has taught software engineering at Carnegie Mellon University. She is a frequently invited speaker at technical conferences and most recently was featured in a television special on object technology aired by the British Broadcasting Company.

Before joining the SEI, Northrop was associated with both the United States Air Force Academy and the State University of New York as a professor of computer science, and with both Eastman Kodak and IBM as a software engineer. As a private consultant, Northrop also worked for an assortment of companies covering a wide range of software systems. She has developed and delivered countless software engineering and object-oriented traning programs and seminars. She is a member of the ACM and the IEEE Computer Society, the Computer Sciences Accreditation Commission, the ACM/IEEE Joint Committee on Software Engineering, and the OOPSLA Organizing Committee, and is the OOPLSA '99 Technical Program Chair.

Lawrence G. Jones is a senior member of the technical staff in the Product Line Systems Program of the SEI. In addition to his product line duties, Jones is also a member of the Capability Maturity Model Integration (CMMI) team. Before joining the SEI, Jones served as principal scientist at the SHAPE Technical Centre in The Hague, Netherlands. He is also the former chair of the Computer Science Department at the U.S. Air Force Academy.

Jones is active in the computing profession and has membership on the ACM Accreditation Committee, IEEE, the Executive Committee of the Computing Sciences Accreditation Commission, and the Colorado SPIN Steering Committee. He holds a PhD in Computer Science from Vanderbilt University and master's and bachelor's degrees in industrial engineering from the University of Arkansas.

Sholom Cohen is a senior member of the technical staff at the SEI and has been at the SEI for more than 10 years. Cohen is a member of the Product Line Systems Program and has authored major technical reports, conference papers on domain analysis and domain engineering methods, and an annotated bibliography of domain analysis. He is a contributor to the Product Line Framework and is also the author of reports on Product Line Concept of Operations for the Air Force Electronic Systems Center, the National Reconnaissance Office, and DoD test and training ranges. Besides domain engineering, Cohen's current research activities include object technology, software product line practices, and product line introduction.

Prior to joining the staff of the SEI, Cohen was a member of the software engineering technology branch of the McDonnell Douglas Astronautics Company. In that position, he was a key developer of the Common Ada Missile Packages components and tools. Cohen received his BS from the Massachusetts Institute of Technology, an MA in Library and Information Science from the University of Michigan, and an MS in computer science from Columbia University.

Dennis Smith is a senior member of the technical staff in the Product Line Systems Program at the SEI. He is the technical lead for the work in reengineering and mining of core assets for product line systems. This work has developed a framework for program understanding, integrated trends toward the use of distributed object technology and Web-based program understanding, and developed an enterprise framework for analyzing reengineering problems.

Before taking on this role, Smith was project leader for the CASE environments project. This project examined the underlying issues of CASE integration, process support for environments, and the adoption of technology. Smith has conducted studies of "lessons learned" from previous efforts at developing large environments and adopting CASE tools. His work includes the development of a CASE adoption strategy and process, a framework for selection of tools and environments, and an analysis of the state of the practice of environment integration.

Smith is a co-author of the book *Principles of CASE Tool Integration*, Oxford University Press, 1994. In addition he has published a number of articles and technical reports, and has presented papers at a number of professional conferences. He is co-editor of the IEEE recommended practice on CASE adoption. Smith is active in both ACM and IEEE and is currently general chair for STEP '99 and for IWPC '99. Smith has an MA and PhD from Princeton University, and a BA from Columbia University.

Paul Clements is a senior member of the technical staff at the Software Engineering Institute. A graduate of the University of North Carolina and the University of Texas, he is a project leader in the SEI's Product Line Systems Program. His work includes collaborating with organizations that are launching product line efforts. He is a co-creator of the Software Architecture Analysis Method (SAAM), which allows organizations to evaluate architectures for fitness of purpose. He and others are working on an extension to SAAM, which will allow analysis of quality attribute trade-offs at the architectural level. He is co-author of *Software Architecture in Practice* (Addison-Wesley-Longman, 1998) and more than three dozen papers and articles about software engineering.

Links to Product Line Resources
References to Product Line Related Readings

The links and bibliography entries below provide information about Product Line Systems research efforts, publications, and general resources.

Links

Product Line Practice Workshop Report
http://www.sei.cmu.edu/publications/documents/97.reports/97tr003/ 97tr003abstract.html

Product Line Case Studies
http://www.sei.cmu.edu/plp/plp_case_studies.html

A Case Study in Successful Product Line Development
http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.016.html

Report of the Reuse and Product Lines Working Group of WISR8
http://www.sei.cmu.edu/publications/documents/97.reports/97sr010/97sr010abstract.html

Software Product Lines: The New Paradigm for the New Millennium
http://www.utexas.edu/coe/sqi

Concept of Operations for the ESC Product Line Approach
http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.018.html

Developing Architecture through Reuse
http://www.sigs.com

A Model for Platform Development

http://www.hp.com

The Economics of Product Line Development

http://www.owego.com/~poulinj

**Software Architectures, Product Lines, and DSSAs:
Choosing the Appropriate Level of Abstraction**

http://www.owego.com/~poulinj

Investment Analysis of Software Assets for Product Lines

http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.010.html

Related Readings

**Process Centered Environments for Software Product Lines:
Requirements and Proposal for Cooperation Support**
Alloui, Ilhquendo, Flavio

(*Proceedings, 10th International Software Process Workshop*, Dijon France,
17-19 June 1996)

Annotation: This four-page position paper for the workshop emphasizes the interactions among participants in large-scale reuse-based projects and the means to facilitate those interactions. The s are with (separate) universities in France.

Supporting Product Line Development
Balzer, Robert

(*Proceedings, 10th International Software Process Workshop*, Dijon France,
17-19 June 1996)

Annotation: This is a short position paper for the workshop that is oriented towards program (application) generators as a key enabling technology for product line development. The is with the Information Sciences Institute.

Exploiting the Synergism Between Product-Line Focus and Software Maturity

Besselman, Joe; Rifkin, Stan

(*Proceedings, 1995 Acquisition Research Symposium*, sponsored by the Deputy Under Secretary of Defense for Acquisition Reform; co-hosted by the Defense Systems Mgt. College and the National Contract Management Association [Washington, D.C. chapter])

Abstract: Using emerging evidence of the benefits of software process improvement, we present a theory for why the software development community is witnessing differential results from its software process improvement efforts. Our theory predicts that firms organized around product lines are more likely to possess higher levels of maturity. Using the results of 51 software evaluations and assessments, a statistical test fails to reject the theory. Our findings lead us to recommend that the government should offer incentives for software process improvement in software intensive procurements, embrace, where appropriate, a product-line approach procurement, and refrain from placing software quality requirements in contracts.

Dividing the Software Pie

Cleaveland, J. Craig; Fertig, Janet A.; Newsome, George W.

(*AT&T Technical Journal*, March/April 1996)

Abstract: Systematic software reuse, or multi-use, is a key to increasing the productivity and quality of software development. In the past 20 years, reuse has experienced many failures and a few successes. Many technological, organizational, and cultural obstacles have been placed in its path. A critical step to increasing software reuse is to recognize that a new division of labor is required, one in which component developers create reusable components and product developers compose products from these components. Changing organizational structure and software development processes to nurture these roles is challenging. Once these roles are recognized and established, however, standard abstraction techniques and other software reuse technologies can help separate the concerns of component developers and product developers. This paper illustrates the separation of concerns by examining its application to interfaces, a particularly difficult area in which these concerns are traditionally intertwined.

Successful Product Line Engineering Requires More Than Code Reuse
Clements, Paul

(*Proceedings, 1997 International Workshop on Software Reuse [WISR8],* Columbus Ohio)

Annotation: This five-page paper is a re-telling of the CelsiusTech experience, and emphasizes that organizational, managerial, process, and cultural issues are all as (if not more) important than technical approaches to reuse in order to successfully deploy a software product line.

Program Families: Some Requirements Issues for Process Languages
Cugola, Gianpaolo; Ghezzi, Carlo

(*Proceedings, 10th International Software Process Workshop*, Ermitage Frere Joseph, Ed., Ventron, France June 17-19, 1996)

Abstract: In this position paper, we address the workshop theme by discussing issues concerning process support for product families. In particular, we discuss how the particular problem of supporting product family developments affects the notation used for process representation and enactment. Based on our current understanding of the problem, our conclusion is that the development of program families does not introduce new requirements, but stress some of the requirements that are intrinsic to most software processes to their extreme. That is, they do not demand new language mechanisms, but require specific processes to be put into place.

Product lines: What are the Issues?
Di Nitto, Elizabeth; Fuggetta, Alfonso

(*Proceedings, 10th International Software Process Workshop*, Ermitage Frere Joseph, Ed., Ventron, France, June 17-19, 1996)

Annotation: This is a short position paper from the referenced conference. The authors point out that information in the literature is quite sketchy about this field. They offer the following two definitions: (1) A product family is "a set of software products, each of them offering the same basic functionality with some significant variations." Example: FrameMaker for different computing platforms. (2) A product line is "a set of software products, each of them offering complementary features, and that are conceived, designed, and implemented to jointly operate in supporting user’s activities." Example: Microsoft Office. All told, this is a nice little paper that lays out some of the conceptual ground and technical areas of interest related to product lines.

Applying Software Product-Line Architecture

Dikel, David; Kane, David; Ornburn, Steve; Loftus, William; Wilson, Jim

(*Computer*, August 1997, pp. 49-55)

Abstract: Software product-line architecture is a powerful way to control the risks and take advantage of the opportunities of complex customer requirements, business constraints, and technology, but its success depends on more than technical excellence.

Applying Domain Analysis and Modeling: An Industrial Experience

France, Robert B.; Horton, Thomas B.

(*Proceedings, Symposium on Software Reuse*, Seattle WA, 1995, pp. 205-214)

Abstract: In this paper we describe our experience in applying domain analysis within a company that develops personal electronic devices. We describe how we tailored the DSSA method to suit our needs and then present the process and representations that we found most useful for this situation. The conclusions and lessons learned are useful because few studies published at this time provide details about applications of domain engineering in commercial development environments.

Notable quote: "An organization that view experience as an organizational asset, rather than as an attribute of an individual, is better able to learn from its collective experiences, and thus is in a better position to improve its process and products." The project described is Motorola’s Paging Products Group (PPG). While this article is mostly about domain analysis and thus not squarely within the scope of a product line bibliography, it does describe building a generic requirements document, clearly an essential task in PL development.

A Meta-Process for the Discovery and Evolution of Software-Reuse Processes

Garg, Pankaj; Jazayeri, Mehdi

(*Proceedings, 10th International Software Process Workshop*, Dijon France, 17-19 1996)

Abstract: New reuse-oriented approaches to software development require software processes different from traditional ones. Our goal is to develop process models for domain-specific kit-based software production and consumption so that an appropriate process model can be packaged as one of the components of the kit. Discovery of process models, especially for nontraditional development methods, is a challenging task. We have explored approaches for: (1) empirically discovering and validating software processes; and (2) monitoring and optimizing (evolving) such a process. In this paper we present an iterative meta-process that represents a first step in this direction. We also report on some early usage of the meta-process for an experimental domain- specific kit production process.

Inside A Hollowed-out Mountain, Software Fiascoes—And A Signal Success

Gibbs, W. Wayt

(News and Analysis, *Scientific American*, August 1997, pp. 33-34)

Abstract: This is a very short, light-weight article about the Cheyenne Mountain Upgrade program. Notable quote: "Perhaps the most important difference between ATAMS and conventional systems is that it will be updated every year, rather than replaced once a decade. And it was designed to be just the first in a product line of related systems. Like a line of car models, its relatives will look and perform differently, but share an underlying design and many of the same innards."

Software Product Lines and their Support by Process Clusters

Gruhn, Volker

(*Proceedings, 10th International Software Process Workshop*, Ermitage Frere Joseph, Ed., Ventron, France, June 17-19, 1996)

Abstract: This position paper discusses software process support for product lines. The underlying assumption is that software products are not developed as isolated entities, but that software products tend to be related. Versions and variants as well as customized subproducts are related to a product kernel. The development of product lines can be supported by clusters of related software processes.

Software Reuse: Architecture, Process and Organization for Business Success

Jacobson, Ivar; Griss, Martin; Jonsson, Patrik

(Addison Wesley Longman; New York, New York, 1997)

Configuring Designs for Reuse
Karhinen, Anssi; Ran, Alexander; Tallgren, Tapio

(*Proceedings 1997 Symposium on Software Reusability*, in *Software Engineering Notes*, vol. 22, no. 3, pp. 199-208)

Abstract: The main problem in developing software product families is how to share effort and reuse parts of design and implementation while providing variation of features and capabilities in the products. We discuss the mechanisms that are commonly used to achieve reuse and sharing in product families, and the kind of variance each is best suited for. Our analysis motivates a need for a new mechanisms to deal with ad hoc variation of features found in different members of a family. We argue that higher level abstraction and parameterization techniques are not well suited for this task. We propose an alternative approach that enables sufficiently detailed designs for every variant and at the same time achieves a level of design reuse without making designs unnecessarily complex or implementations inefficient.

Integrating and Applying Processes and Methods for Product Line Management
Klingler, Carol Diane; Creps, Richard

(*Proceedings, 10th International Software Process Workshop*, Dijon France, 17-19 June 1996)

Annotation: This short position paper details an "integrated set of concepts, processes, methods, and tools, called ReuseWorks, which supports the product line approach...and has been validated through usage on the Army START Demonstration Project."

Creating Reusable Architectures: Initial Experience Report
Lam, W.

(*Software Engineering Notes*, vol. 22, no. 4, July 1997, pp. 39-43)

Abstract: Achieving systematic reuse of software designs requires the creation of a reusable software architecture. This paper describes initial experience of creating reusable architectures in the avionics domain. A case-study is presented which illustrates the RACE (Reusable Architecture Creation and Employment) process. In RACE, a variability analysis is used to identify possible variations in a family of systems, which then guides the creation of a generic architecture and a set of architectural "plug-ins." This paper concludes with a set of RACE guidelines which summarizes our initial experience.

Managing Domain-Specific Product-Line Development
Macala, Randall R.; Stuckey, Lynn D.; Gross, David C.

(*IEEE Software*, May 1996, pp. 57-67)

Abstract: A $14 million demonstration project revealed that product-line development requires a mature process and a sophisticated support environment. The results indicate that it will demonstrate a return on investment for lines of three or more systems.

The Product Family and the Dynamics of Core Capability
Meyer, Marc H.; Utterback, James M.

(*Sloan Management Review* 34, [Spring 1993]: 29-47)

Abstract: Individual products are the offspring of product platforms that are enhanced over time. Product families and their successive platforms are themselves the applied result of a firm's underlying core capabilities. In well-managed firms, such core capabilities tend to be of much longer duration and broader scope than single product families or individual products. The authors recommend a longer run focus on enhancing core capabilities, which includes identifying what they are and how they are applied and synthesized in new products.

The Design and Development of Information Products
Meyer, Mark H.; Zack, Michael H.

(*Sloan Management Review*, Spring 1996, pp. 43-58)

Abstract: Companies that produce information in printed or electronic form can learn much from research on physical products, including the development of product and process platforms to enhance design and development. The authors provide a framework for the architecture of information products and apply it to two companies that are creating competitive advantage by refining information through product and process technologies. The authors also consider ways that companies can design information products in the future to focus on customers' implied needs and to take advantage of new interactive technologies.

Notable quote: "Every product has an architecture. That architecture has the potential to become a platform... Rather than product architecture constraining product variety, Black & Decker's common subsystems enabled variety, allowing engineers to focus on the other features of their power tools, such as new types of bits, sanding surfaces, or blades. Product variety increased over time. The company's cost advantage drove dozens of competitors out of business..."

Metrics for Managing Research and Development in the Context of the Product Family
Meyer, Mark H.

(*Management Science*, 1997)

Abstract: The paper proposes methods to measure the research and development in new product development. We frame these measures in the context of evolving product families in the technology-based firm. Our goal is to more clearly understand the dynamics of platform renewal and derivative product generation and their consequences for long term success. We explore the utility of the proposed methods with data gathered from a large measurement systems manufacturer. We find that the methods and measures can help management assess the technological and market leverage achieved from the firm’s present and past product platforms. This provides a foundation for transforming single product, single period planning processes into a multi-product, multi-period form that embraces the product family and the renewal of product architecture. The research also shows the need to integrate data from engineering, manufacturing, and sales organizations to product information for managing the growth of the firm’s product families.

Annotation: A thorough and scholarly treatment of product family metrics at the organizational level. Manuscript is approximately 40 pages. Notable quote: "When should a firm renew the underlying technologies and designs of its products? How much will these efforts cost and how long may they be expected to take? What types of engineering and commercial benefits can the firm expect to gain from product redesign? How can a firm improve its approaches and strategies for product development?"

On the Design and Development of Program Families
Parnas, D. L.

(*IEEE Transactions on Software Engineering*, SE-2, 1, pp.1-8, 1976)

Annotation: This is a foundation work of the field, and one of Parnas’s more important papers. In it, he shows how program development is essentially a path down a decision tree, with each node corresponding to a design decision. Decisions towards the top of the tree are the hardest to change (because they require more back-tracking), whereas decisions near the leaves of the tree are much easier to change. The leaves of the tree form a program family, which Parnas defines as a set of programs for which it is useful to consider as a group rather than individually. The moral of the paper is that it pays handsomely to pay careful attention to the order in which design decisions are made, deferring the ones most likely to change until the end.

Experiences Developing and Maintaining Software in a Multi-Platform Environment
Pearse, T. Troy; Oman, Paul W.

(*Proceedings, International Conference on Software Maintenance*, Bari Italy September/October 1997)

Abstract: The computer market demands that companies develop families of software products that can be scale to meet the functional and performance needs of the personal and business computer markets. To support a family of LaserJet printer products, Hewlett-Packard defined the multi-platform parallel development model for software development. This model allows HP to simultaneously develop a family of LaserJet printers that have different features and run on different processors, while shortening the development. In this paper we discuss our experiences using a technique called conditional compilation, within the multi-platform parallel development model, to create portable, scaleable software systems. We describe and share a new tool that was developed to help understand code containing conditional compilation. Examples of using the tool on industrial source code, and lessons learned while managing conditional compilation complexity, are provided.

Product-Line Reuse Delivers a System for One-Fifth the Cost in One-Half the Time
Randall, Richard L.; Bristow, David; Foster, Jesse; Kaip, Dennis

(*Crosstalk: The Journal of Defense Software Engineering,* August 1996, pp. 25-26)

Abstract: This article summarizes the Automated Tracking and Monitoring System (ATAMS) project and technologies the team integrated to make the project a resounding success; it tells why the Space and Warning Systems Center believes the resulting product-line approach is a promising metaphor to produce and maintain its family of software-intensive systems.

Reuse-Driven Software Processes Guidebook
Software Productivity Consortium

(Software Productivity Consortium report, SPC-92019-CMC, Version 02.00.03, 1993. Available through the Software Productivity Consortium, Herndon, Virginia)

Reuse Contracts: Managing the Evolution of Reusable Assets
Steyaert, Patrick; Lucas, Carine; Mens, Kim; D'Hondt, Theo

(*Proceedings, OOPSLA 1996*)

Abstract: A critical concern in the reuse of software is the propagation of changes made to reusable artifacts. Without techniques to manage these changes, multiple versions of these artifacts will propagate through different systems and reusers will not be able to benefit from improvements to the original artifact. We propose to codify the management of change in a software system by means of reuse contracts that record the protocol between managers and users of a reusable asset. Just as real world contracts can be extended, amended, and customized, reuse contracts are subject to parallel changes encoded by formal reuse operators: extension, refinement, and concretization. Reuse contracts and their operators serve as structured documentation and facilitate the propagation of changes to reusable assets by indicating how much work is needed to update previously built applications, where and how to test and how to adjust these applications.

Software Asset Management and Domain Engineering

Subramanian, Satish

(*Proceedings, 21st. Annual International Computer Software and Applications Conference [COMPSAC]*, 1997.)

Annotation: This is a short (two-page) position paper for a panel presentation. The opening text follows: "Promoting reusability by managing software assets can greatly benefit companies that develop a family of similar products, where products are evolving from another. One of the main goals of domain engineering is to identify and document the commonalities across the various products in a particular domain... Guidant corporation has been involved in the development of medical devices... These devices and related products are constantly evolving as technology and market needs change. The systems being developed... at thus a family of products and share many functionalities among them."

Product Families and Process Families

Sutton, Stanley M. Jr.; Osterweil, Leon J.

(*Proceedings, 10th International Software Process Workshop*, Dijon France, 17-19 1996)

Notable quote: "Our work in software processes has led us to conclude that software processes also can be usefully viewed in terms of families."

The Architect

The Perils and Joys of Reconstructing Architectures

Steven G. Woods, S. Jeromy Carrière, Rick Kazman

A documented, analyzed software architecture is a key ingredient in achieving quality in large software-intensive systems. But the system that is implemented must conform to its architecture for the qualities of the design to carry over into the implementation. To ensure that systems conform to their architectures, and remain in conformance throughout their lifetimes, we need to reconstruct architecture from source artifacts. To do this properly, a wide variety of tools that provide both static and dynamic information are needed. Thus, we advocate a *workbench* approach to architecture reconstruction tools.

Why Reconstruct Software Architectures?

Evaluation of an architecture's properties is critical to successful system development [1]. However, reasoning about a system's *intended* architecture must be recognized as distinct from reasoning about its *realized* architecture. As design and eventually implementation of an architecture proceed, faithfulness to the principles of the intended architecture is not always easy to achieve. This is particularly true in cases where the intended architecture is not completely specified, documented or disseminated to all of the project members. In our experience this is the rule, and well-specified, documented, disseminated, and controlled architectures are the exception.

This problem is exacerbated during maintenance and evolutionary development, as architectural drift and erosion occur. However, if we wish to transfer our reasoning about the properties of a system's intended architecture to the properties of the implemented system, we must understand to what degree the realized architecture *conforms* to the intended architecture.

Architectural conformance may only be measured if we have available two architectures to compare: the intended architecture and the architecture as it is realized in the implemented system. The former should be documented early in a system's lifetime and maintained throughout. The latter, however, typically exists only in artifacts such as source code and makefiles and, occasionally, designs that are directly realized as code (through, for example, the use of an architecture description language). In addition, it is infrequent that an implementation language provides explicit mechanisms for the representation of architectural constructs. Therefore, facilities for the *reconstruction* of a software architecture from these artifacts are critical in measuring architectural conformance.

Beyond its importance for measuring architectural conformance, software architecture reconstruction also provides important leverage for the effective reuse of software assets. The ability to identify the architecture of an existing system that has successfully met its quality goals fosters reuse of the architecture in systems with similar goals; hence architectural reuse is the cornerstone practice of product line development.

In the remainder of this paper we will discuss several issues relevant to the successful reconstruction of software architectures. These issues include the following:

- the need to expand our horizons beyond the use of purely static information during reconstruction
- the need to support the reconstruction of software architectures based on a paucity of information, such as non-compilable code, obsolete artifacts, or the absence of architectural information
- the need to leverage existing commercial and research tools within a comprehensive reconstruction framework

Static Information Is Insufficient

A significant quantity of information may be extracted from the static artifacts of software systems, such as source code, makefiles, and design models, using techniques that include parsing and lexical analysis. Unfortunately, system models extracted using these techniques provide a minimum of information to describe the run-time nature of the system. The primary factor contributing to this deficiency is the widespread use of programming language features, operating system primitives, and middleware functionality that allow the specification of many aspects of the system's topology to be deferred until run-time. All but the simplest systems use some subset of

- language features such as polymorphism and first-class functions (including approximations such as those provided by C and C++)
- operating system features such as proprietary socket-based communication and message passing
- middleware layers such as CORBA

These mechanisms permit systems to be designed with low coupling and a high degree of flexibility. While these are laudable goals, they obscure the architecture reconstruction process.

In particular static extraction techniques can provide only limited insight into the run-time nature of systems constructed using such techniques, because many of the details that determine actual communication and control relationships simply do not exist until run-time, and hence cannot be recognized until run-time. For example, relationships among communicating processes might be determined via an initialization file, or even dynamically, based upon the availability of processing resources. However, most existing architecture reconstruction tools depend almost exclusively on abstract syntax trees (ASTs) extracted using parsing-based (i.e., compile time) approaches, and so they cannot gather such information.

To achieve a better understanding of the run-time nature of systems that leverage mechanisms such as those described above, we must consider how we may go about extracting dynamic information from a running system. Some techniques to accomplish this include profiling and user-defined instrumentation. Profiling is a technique that is traditionally used for system performance analysis. When profiling, one typically compiles a system with a special flag that instructs the compiler to instrument the code such that it records information pertaining to function invocation during execution. The system is then exercised and the recorded information is analyzed. We may use this technique to determine actual function invocations, augmenting our statically extracted models with improved information concerning polymorphic functions and functions executed through function pointers (e.g., in C or C++).

In a similar fashion, user-defined instrumentation is a technique for adding special-purpose tracing functionality to a system to allow monitoring of its operation. For example, instrumentation can be added to application code responsible for interprocess communication to determine the system’s run-time communication topology. In addition, it is sometimes possible to instrument libraries or even the operating system. This allows the possibility of instrumentation of systems without modifying any application code and requires less application-specific knowledge.

Non-Compilable Code

Throughout our experience with software architecture reconstruction, we have frequently been faced with systems that cannot be compiled. Often we are provided with a complete body of application code, but we are missing the header files and/or some of the libraries that are needed to compile it. In other cases, the application code is written in a peculiar dialect of a standard language (e.g., Objective C or an obscure dialect of Fortran), or implemented on an uncommon hardware platform, and thus may only be compiled with specialized tools that are not available to us. Off-the-shelf parsers and analyzers simply do not apply in these cases. There are cases when this lack of information makes a tool-facilitated reconstruction of the architecture nearly impossible. For example, when dynamic binding of function calls or dynamic relationships between processes are used extensively in a system, we may need to compile and run the system to determine the true relationships between the components.

Fortunately, we can often reconstruct software architectures even under these difficult circumstances. To accommodate non-compilable code whenever possible, we can avoid complete reliance on parsing-based techniques and can resort to lexical analysis for information extraction. While this is not an ideal situation (you often don't have as much information to work with in these circumstances, and lexical techniques are frequently unreliable [2]), it is reality.

Reconstructing Architecture in a Vacuum

Unfortunately, it is frequently the case that our efforts to reconstruct the software architectures of systems must contend with a complete lack of pre-existing architectural information. This often occurs when the system being analyzed is particularly old (and thus there are no longer any designers or developers who can relate architectural information) or when the system's intended architecture was never documented. Furthermore, these are typically the situations in which we are most interested in recovering a system's architecture. In particular, it is common for such systems to be involved in ongoing maintenance or even undergoing a more global re-engineering effort (e.g., modernization or porting).

Successful architecture reconstruction revolves around the acquisition of architectural constraints and patterns that capture the fundamental elements of the architecture. Regardless of the mode of development of a system, its evolutionary state, or its age, such constraints and patterns are always present. However, they are rarely (even when a truly architecture-based development process is followed) captured explicitly. Thus, the primary task of the reconstruction analyst is acquisition of this information by means other than investigation of documentation.

One mechanism for acquiring architectural patterns is exploration of the information extracted from the system artifacts. Such exploration will often uncover frequently-used idioms of control flow, data flow, interface usage or interaction among the architectural components of the system. An alternative to this type of ad hoc exploration is the application of an architecture analysis method, such as the Architecture Tradeoff Analysis Method (ATAM) [3], as an elicitation device. Although ATAM was developed as a method for analyzing the tradeoffs inherent in architectural design, it may also serve as a structured way to elicit architectural information. ATAM is scenario-based: Scenarios are used to capture uses of and changes to the system being analyzed. It is the “scenario mapping” step of the method that is useful for architectural elicitation; in this step, the scenarios are traced through the architecture. For uses of the system, the participating components and their communication patterns are identified. For changes to the system, the architecture-level impacts are identified. In this way, we have a structured exercise for exploration of the architecture of the system. If the system being analyzed is indeed undergoing some level of maintenance, there will always be one or more developers who have some knowledge of the system’s operation and can participate in such an exercise.

An Architecture Reconstruction Workbench

Based upon the above observations, in developing tool support for software architecture reconstruction, our realization was that no particular static set of tools (extractors, visualization tools, analysis tools, etc.) would suffice for every task. At the very least, we wanted to support many implementation languages, many target execution platforms, and many techniques for architecture analysis. Thus, we wanted a *workbench*: an *open* and *lightweight* environment that provides an infrastructure for the opportunistic integration of a wide variety of tools and techniques. The workbench must be open to allow easy integration of additional tools. The workbench must be lightweight to ensure that no unnecessary dependencies exist among already integrated tools and data.

We realized the workbench concept in Dali, a support environment for software architecture reconstruction [4]. Dali’s current implementation uses an SQL repository for data storage and manipulation. SQL provides the medium for the primary activities of the Dali user: architectural pattern definition, view creation, view fusion, and pattern recognition.

View fusion is a technique for combining extracted information—views—from different sources (such as code artifacts, directory structures, naming conventions, execution information) to improve the quality of the overall model of the system [5]. Architectural patterns are the medium for a Dali user to express an understanding of a system’s architecture as structural and attribute-based relationships among its components. The patterns are most commonly expressed via SQL, but may, in principle, be expressed via any of the tools that Dali integrates.

Rigi [6], which provides flexible graph editing functionality and effective end-user programmability, supplies Dali’s user interaction and also acts as a vehicle for tool integration.

Dali, as represented in Figure 1, currently integrates several commercial and research tools for the following:

- extraction of information—rigiparse for C, Imagix [7] for C and C++, SNiFF+ [8] for C++ and Fortran, LSME [2] for C++
- visualization—dot for graph layout
- architectural analysis—IAPR [9] for architectural complexity analysis, RMTool [10] for automatic conformance measurement

The use of commercial tools has provided a great deal of leverage during the development of Dali: These tools are typically robust, well documented, and well tested.

Dali offers an iterative, interpretive, user-driven approach to architectural reconstruction. Our view is that no system has "an" architecture. It has many views of a complex body of interrelated information and the choice of which views to extract and reconstruct is driven by the user's information needs. It is, however, interesting to consider the implications of the user-driven nature of Dali with respect to the issues raised above.

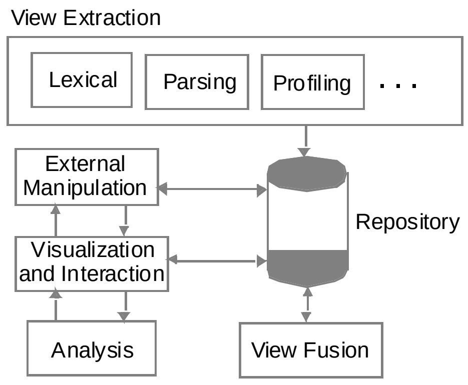


Figure 1: The Dali Workbench

Consider the case of non-compilable code: Because of the iterative nature of Dali, we can use the information gleaned from purely lexical extractors to guide the user's investigations. For example, in this case the user could extract some portion of the architectural information— knowing that it was incomplete—and then reconstruct just part of the architecture. This would then lead to new questions being asked and new lexical extractors written, and the process repeats with increasing fidelity.

In a similar fashion, when we extract views from dynamic information, these views are incomplete, since dynamic information, like run-time testing, only reflects the parts of the system exercised during the execution. To gain a complete picture of the system, run-time views need to be fused with other (possibly static) views to improve the overall quality of the reconstruction.

Lastly, it is again Dali's iterative and interactive nature that makes it appropriate for supporting a process of reconstructing architecture in the absence of significant architectural information. Lightweight view extraction and flexible visualization provide an environment that fosters opportunistic exploration of the available information. These features also provide the necessary support for the scenario-mapping task that is central to the application of the ATAM method in this context.

It is the combination of Dali's openness and its interactivity that helps us overcome the inherent limitations of any single extraction and reconstruction technique. Dali does not offer a monolithic solution, but rather a set of tools, some of which are heuristic. As a result, we have been able to apply Dali to a wide range of systems, written in many different languages and dialects, written for different hardware platforms and operating systems, and created with vastly different levels of architectural expertise.

Architecture Reconstruction in Practice

As part of a large-scale product-line migration effort, the Software Engineering Institute is assisting a major U.S. industrial organization ("BigTruck Inc.") in the identification of key architectural elements and their relationships in an existing real-time embedded control system called "X1." The existing legacy system supports many product instantiations. X1 is being migrated to a product line, "X2." The set of X1 components will be identified as an "X1 baseline," to identify the current operating architecture of the in-place software. It is expected that this baseline model will provide the starting point for trading off a variety of to-be-determined architectural requirements derived from existing X1 customers and BigTruck sources as well as new or changed requirements for the future X2 (and beyond) architectures. The key players in this effort are the SEI team members and the newly formed BigTruck Software Architecture Group (SAG) headed by a very competent architect, "Hugh Effert."

The SEI team and the BigTruck architect have worked together to define a reconstruction process to support the product line migration. This process was evolved in an effort to assist the SAG in gaining control over the many X1 components—control that was initially distributed among a number of orthogonal sub-projects. Further, the process is intended to support the reality that migration to product lines must usually accommodate moving-target baseline systems. In the case of BigTruck, three individual business-critical deliveries of instantiations of the X1 architecture will occur prior to the release of X2.

The process of reconstructing the X1 architecture currently underway at BigTruck is summarized in the major steps outlined below.

Task A: Develop Component Information/Identification Form

In order to reconstruct an architectural representation it is first necessary to identify what the core set of components is in the particular implementation. This identification form should provide examples and guidance to system experts to identify “what makes a component a component” in the particular system or sub-system. The form will be used in Task C when SAG members formalize the properties of components initially identified by the SAG in Task B. The form is meant to provide guidance to SAG members as they attempt to carefully specify what attributes and relationships in the implementation signify membership in (or exclusion from) a particular component. Examples of identifying attributes include naming conventions for code elements such as functions or variables and code location in terms of directories. Examples of relationships may include certain control-passing styles among components or within a particular component, or potentially the types and names of data used within a particular component.

Task B: Identify Components and Layers

As a first step, the SAG will attempt to determine major responsibilities and architectural layout by naming the major architecture components and attempting to assign them to layers according to current beliefs (hopes?) about the layered structure of the existing X1 code base. Certainly it is currently believed that the X1/X2 (new) architecture is intended to reflect the layers represented as closely as is practicable. Following completion of this initial SAG effort, the individual components will be characterized according to properties laid out in Task C.

Task C: Populate Information Forms

Using the forms created in Task A, the SAG member responsible for each component will specify the defining characteristics of the component. This process of expert identification of architectural signatures is critical to the task of extracting the “as-implemented” architecture as this information is the primary source for the specification, in Task D, of rules for clustering software into architectural components. The process of populating the form with information developed in Task A may result in the modification of the form itself as unanticipated features characterizing architectural components are discovered by SAG members.

Task D: Develop Architectural Rules

From the information forms filled in by the SAG group members, the SEI will work with Hugh to represent the information describing the architectural components in Dali SQL queries, which will be used to match (Task E) the components from the extracted system information database built in Task F. The SEI team can commence planning the rule descriptions as soon as the information forms begin arriving from the SAG team members.

Task E: System Extraction to Database

The SEI will, once the BigTruck source code and associated makefiles have been received, parse the system into an intermediate representation stored in a Dali relational database. While there is a standard set of parsing tools the SEI typically uses in such extractions, the complexity of the identifying features used in component description in Task C and D determines the degree of detail required in the parse-and-represent process.

Task F: Apply Rules to Database

Following the population of the system database and the development of the architectural rules (Tasks D and E), the SEI team will utilize Dali to match the rules against the extracted system. The resulting clusters can be visualized and further manipulated using Dali. This work will be done in conjunction with the SAG in order to facilitate the transfer of the skills required to BigTruck in a timely manner. The actual rule application (and possibly refinement) may determine new constraints that require additional (or repeated) extraction work, possibly with additional tools, and may also impact the rules defined and potentially even the nature of the information forms used to determine the component identity features.

Task G: Map Models to Architectural Views

Using Dali's query, visualization, and manipulation tools, the extracted models of the X1 system will be mapped to a standard UML template (previously developed by the SAG). The architectural views will be constructed showing the "as-implemented" architecture given the component definitions and rules developed earlier. The mapping process will be undertaken jointly between the SEI and BigTruck's Hugh Effort.

Task H: Evaluate Architectural Views

The views built during task G will be evaluated for use in the ongoing development of the X2 architecture. This evaluation will verify that the views are able to support the types of design and analysis that the X2 architecture development process will require. The sub-processes contained from Tasks D, F, E, and G are presumed to be iterative as required.

The BigTruck Big Picture

The BigTruck effort is a tremendous example of the role software architecture reconstruction can (and should) play in the context of a product-line migration. The migration itself depends upon reliable views of the existing software architecture in addition to a clear understanding of the new architecture. The SEI has assisted BigTruck in generating its new X2 architecture by mapping BigTruck's new requirements against its existing X1 architecture representation. Evaluation of the new architecture is being conducted through architectural reviews and the use of the SEI's Architecture Tradeoff Analysis Method [3]. Migration plans are being made in an incremental manner, allowing BigTruck to continue delivering X1 instantiations during the X2 evolution. Ultimately, the mapping at the code level to the X1 architectural representation will support a reliable source migration to the X2 architecture. Since BigTruck plans to generate performance models from its new X2 architecture and make performance tradeoff decisions based on these models, the accuracy of the X2-to-source mappings must be trustworthy.

About the Authors

Steven Woods is a Member of the Technical Staff at Carnegie Mellon University's Software Engineering Institute, where he is a member of both the Reengineering Center and the Architectural Tradeoff Analysis (ATA) Initiative. The Reengineering Center has a mandate to identify, enhance, and transition best practice in software reengineering in a disciplined manner, while the goals of the ATA Initiative are to refine the set of techniques for analyzing architectures with respect to various quality attributes, to develop a method for understanding how these attributes interact, and to offer the software development community mature architecture evaluation practices.

Steven's particular expertise lies with analyzing, applying, integrating and extending existing and emerging toolsets for software reengineering, understanding and architectural analysis. In particular, the SEI wishes to leverage these tools as an crucial part of the process of developing and evolving software systems into product-line assets.

S. Jeromy Carrière is a member of the technical staff at the SEI. Before joining the SEI, Carriere was a software engineer with Nortel (Northern Telecom). His primary interests are related to software architecture: recovery, re-engineering, representation, analysis, and tool support. He is the author of several papers in software engineering, information visualization, and computer graphics. Carriere received a B. Math from the University of Waterloo and is a member of the Association for Computing Machinery.

Rick Kazman is a senior member of the technical staff at the SEI, where he is a technical lead in the Architecture Tradeoff Analysis Initiative. He is also an adjunct professor at the Universities of Waterloo and Toronto. His primary research interests within software engineering are software architecture, design tools, and software visualization. He is the author of more than 50 papers and co-author of several books, including a book recently published by Addison-Wesley entitled *Software Architecture in Practice*. Kazman received a BA and MMath from the University of Waterloo, an MA from York University, and a PhD from Carnegie Mellon University.

References

[1] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1997.

[2] G. Murphy, D. Notkin, “Lightweight Lexical Source Model Extraction,” *ACM Transactions on Software Engineering and Methodology*, 5(3) (July 1996), 262-292.

[3] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, S. J. Carrière, “The Architecture Tradeoff Analysis Method,” *Proceedings of ICECCS*, (Monterey, CA, July 1998).

[4] R. Kazman, S. J. Carrière, “Playing Detective: Reconstructing Software Architecture from Available Evidence,” *Journal of Automated Software Engineering*, 6(2) (April 1999), 107-138.

[5] R. Kazman, S. J. Carrière, “View Extraction and View Fusion in Architectural Understanding,” *Proceedings of the 5th International Conference on Software Reuse* (Victoria, BC, Canada, June 1998), 290-299.

[6] K. Wong, S. Tilley, H. Müller, M. Storey. “Programmable Reverse Engineering,” *International Journal of Software Engineering and Knowledge Engineering*, 4(4) (December 1994), 501-520.

[7] Imagix Corporation, http://www.imagix.com.

[8] TakeFive Software, http://www.takefive.com.

[9] R. Kazman, M. Burth, “Assessing Architectural Complexity,” *Proceedings of 2nd Euromicro Working Conference on Software Maintenance and Reengineering*, (Florence, Italy, March 1998), 104-112.

[10] G. Murphy, D. Notkin, K. Sullivan, “Software Reflexion Models: Bridging the Gap between Source and High-Level Models,” *Proceedings of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering* (Washington, D.C., October 1995).

The Business Edge

Eight Key Factors for Successful Technology Collaboration

Mike Mattison

Many companies today are involved in various forms of partnerships, alliances, and collaborations. “Technology collaborations” are formed specifically to improve the software development organization, in whatever capacity, whether it involves technical methods, processes, or practices. In all cases, these collaborations are difficult to manage, control, and track to productive closure. To help guide the collaboration process, it is necessary to first develop your organization’s software strategy. In this, the second of two columns on the subject, I will explore in detail the eight key factors that I consider essential to developing an effective software strategy as the basis for managing and guiding the technology collaboration activity.

Which Attributes Are Vital?

Companies and their markets have many variant business drivers and operating conditions, and a “universal strategy” cannot be expected to address software engineering issues across all conditions. But which attributes, or key factors, are vital to managing the technology collaboration process? How can we better understand which factors are the vital core to building an effective software strategy?

By using the following eight key factors as a baseline analysis, organizations can formulate an effective strategy in support of complex technology collaborations.

1. Technical Issues

Organizations must define and understand their primary software technical issues. This assumes that software is an essential component of your ability to produce and deliver value to your customers. Generally, this requires your team to analyze all key factors that impact your technical capacity or abilities to develop software. Here are some questions to consider in capturing this in your strategy:

- What are at least three technical issues perceived as critical to improving or evolving your software engineering capabilities? Why are they critical?

- What types of analyses or diagnostic methods do you employ to evaluate your software development capabilities and improvement strategies?
- What types of measures apply to software development, and how is data captured, recorded, and used in decision analysis?
- How do you rank criticality of such factors as cost, schedule, performance, safety, reliability, or time to market in assessing your software performance or improvement analysis?
- What is the greatest barrier to improving how your firm deals with primary software technical issues (e.g., training budget, management support, etc.)?

2. Primary Business Drivers

Your organization must understand the key factors that drive your business. Software is not developed in a vacuum; it is a major element of competitive advantage. The best-performing firms have already closely aligned their software development capability to the core business of the firm. For instance, embedded software that is developed based on accurate market research can result in feature-rich products that customers value. Manufacturing systems don't just coordinate plant-floor mechanical production, but now also gather key data into management reports and analyze that data in ways that facilitate rapid decision making and course corrections.

Clearly, a software strategy that is disconnected from the core business drivers will fail. In the final analysis, the software strategy must consider the primary business drivers of the industry, and perhaps the customer markets. Often these drivers will have the greatest impact on your success, and perhaps on your customers' success. For instance, firms that rely on critical manufacturing cycles (e.g., automotive and office-equipment firms) may be driven by product schedule, and cycle time may be the business driver for the embedded software. Thus, a strategy of software process discipline becomes vital to developing schedule-release integrity. Another example is firms that rely on critical safety or reliability (e.g., firms in the airline or currency-trading industries) in which large-scale, mission-critical systems are subject to extreme quality and safety compliance as the business driver. In such cases, a strategy of architecture product configurations may improve technical control over software upgrades, technology insertion, and system-performance enhancements.

In the SEI's Strategy Workshop, we ask executives and managers to make a list of their primary business drivers. These might include factors such as manufacturing schedule, user quality requirements, government safety regulations, cost, or feature expansion. One commonly occurring problem is that management teams have different interpretations of business drivers, but also have divergent expectations for how software should contribute to their business goals. (For more information on the SEI's Strategy Workshops, please contact me at mvm@sei.cmu.edu or 412-268-3628.) Some questions to consider are:

- What is your core business, and why do customers buy your product or service? What is the "value proposition" that your product fulfills?
- What are the major business drivers of your industry, and how does software contribute to achieving these drivers for your firm? For example, a major driver in the automotive industry is safety, and embedded electronics software is improving component integrity and reliability on such components as anti-lock brakes, engine controls, and driver displays.
- How does technological change or process evolution impact your ability to deliver leading software performance, quality, and cost management to your customers?
- How much of your firm's marketing strategy is derived from features that are created by software? That is, how does software contribute to creating or enhancing the most valued attributes of your product or service?
- How much of the total cost of your product or service is from software, and what is that cost trend?
- What is the greatest barrier to improving how your firm relates software to the key business drivers?

3. Improvement History

Nothing is as ineffective as launching an improvement program that already failed in the past. Too often, organizations do not evaluate their improvement history as part of their efforts to create successful change programs. Before you launch any form of software improvement effort, know your organization's history of improvement. Search for advice from a variety of professionals in your company, and learn what has been tried in the past, what worked, and what failed. Your software strategy should demonstrate an understanding of the firm's history of improvement, the culture of improvement decision making, and how resources are allocated in your organization for software improvements.

Understand how software improvements have been historically managed, and define the resulting impacts upon your software operations and business. Who defines new software technology programs or improvements, the technical organization or management? How well have change efforts been accepted across your organization, and then implemented? Do you track performance and outcomes of your improvement programs? It helps to consider how your firm attends to related activities. These may include using incentives for change, managing data or information to support the need for change, developing executive buy-in before proceeding, and negotiating resource decisions to support new improvements. Some questions to consider in capturing this in your strategy are:

- What has been the most successful improvement effort for your software development organization? What has been the least successful?
- Who is responsible for software, and what is the relation of the responsible party to other business units—for example the design, production, marketing units?
- Have past efforts produced changes in the organization, structure, teams, process, or training?
- What is the history of Total Quality Management (TQM) in the software organization?
- Can you provide a brief explanation for how training supports continuous improvements or new programs for software engineering?
- What is the greatest barrier to creating positive changes in the design, development, and implementation of your software improvement programs?

4. Performance Goals

How does your organization define, manage, and improve software performance? This area of software strategy is among the most difficult to manage, but may produce the greatest near-term gains for your organization. Certainly a strategy cannot be achieved if there is no consensus for performance goals and outcomes. For instance, a strategy to increase sales by 50 percent is not attainable if production performance is already at 96 percent capacity. Understanding performance goals is especially important for evaluating and designing technology collaborations that are intended to impact the software engineering operation.

We have observed numerous firms that have invested in various new technologies without first achieving agreement about their software performance goals. If you presume that new technology will improve some aspect of your software capability, it is essential to define software performance in order to gauge the impact of any new technology. The matter of software performance nearly always produces some volatile exchanges among managers. The fact is, many organizations have not defined "performance" for their software organization. By contrast, manufacturing performance and marketing performance are typically defined by goals, dates, percentage deltas, volumes, and other reliable metrics which, when combined, yield some desired performance indication on a routine basis. A more difficult analysis is to define how your software performance relates to your business performance. For example, as automotive sub-assemblies (e.g., anti-lock brakes, engines, and interior electronics) become increasingly software-intensive, manufacturers will want to demonstrate how increased feature-performance (derived from software) relates to business metrics such as increased unit sales, new vehicle contracts, and improved customer satisfaction. Questions to consider include:

- How does your software team define software performance? What are the performance goals for software in your firm?
- How do your customers define software performance? How well does this definition (or expectation) align with your own definition?
- What measures are used to track software performance goals? What is the quantitative basis for tracking and managing this performance?
- How are software goals tied to business goals? Who owns the decision process?
- What level of executive sponsorship is required to support new, evolving goals?
- What role does software engineering play in the development of these goals?
- How does software performance fit into your strategy?

5. Executive Owner

In every company we know of, senior executives are concerned with efforts that will exceed one year in length, consume numerous company professionals, and require budgets in excess of $50,000. Your software strategy must outline the role and involvement of key senior executives. These executives are required as “sponsors” for the technology collaboration. These executive sponsors often assume long-term roles that take time and effort to ensure high-level awareness, buy-in, understanding, and approvals for the ongoing collaboration. Some software groups operate with great autonomy, making technical and business decisions without significant executive oversight. Other firms require executive involvement and support to ensure successful outcomes to changes in the software organization. In either case, your software strategy must define executive-level sponsorship for the collaboration. Some questions to consider are:

- Who manages the software engineering organization at your firm? Map out the management hierarchy and examine the various reporting relationships up to the executive team.

- What role does executive management have in relation to your software organization? Can your firm expect successful change without senior management's involvement?

- Who is responsible for software at your firm? What is that party's interaction with senior management?

- How has software performance influenced your products, services, and internal operations?

- Who is responsible for bridging software technology to business operations?

6. Management Team

How is management structured in relation to your firm's software development operations? In nearly all cases, successful collaborations are the result of widespread support across management teams. For example, the software strategy for the cell phone division of a telecommunications firm references managers from the marketing, product development, manufacturing, and packaging groups. As software becomes increasingly critical to the core business of the firm, it is important to involve key members of the management team in developing and endorsing the software strategy. This is especially useful when new collaboration opportunities emerge, as the collaboration may depend upon proactive support—and perhaps resources—from those managers who stand to benefit from improved software performance and features. Consider whether your software teams are centralized or distributed across your operations. Does your firm have a CIO who directs software strategy, or do you have several distinct areas that own their own software functions? In many cases, we have seen the value of mapping out how management controls the software assets of the firm. This helps the organization develop a strategy that includes key managers, and aids in the analysis of critical areas for change or improvement. Some questions to consider in capturing this in your strategy are:

- What would a map outline of all key managers involved with software look like? (Include all managers in development phases, product design, marketing, service, etc.)

- To what extent should you involve all managers in the software strategy? Why?
- How vital are software performance, quality, cost, and schedule to your management team? How are these items measured, and how are the people responsible for them rewarded?
- How does management measure software activities? How are projects tracked and managed?
- Are your customers involved in software program activities? Do your customers and users take interest in your software technology programs and innovations?
- What do managers do to gain buy-in and support from senior executives?
- Which managers are vital to the long-term support of new technology collaborations to improve software?

7. Software Engineering

What are your firm's software engineering capabilities, technical processes and methods, and overall engineering conditions, resources, platforms, and operations? The selection and design of effective technology collaborations requires a comprehensive analysis and understanding of your software engineering capabilities. We are often surprised by the lack of internal evaluation by corporate software groups of their engineering resources and capabilities. Internal evaluations are necessary to formulate defensible strategies and plans for technology improvements. Clearly, your capacity to engage new software technologies is a function of how well you understand your existing capabilities and competencies. This is precisely the purpose of the software strategy: to lay down a solid foundation of principles and actions, and to balance those attributes that are essential to advancing our ability to achieve the core goals of building great software. In evaluating your software engineering capabilities, also consider factors such as technical training, resource allocation, and measurement. These also may have an impact on your software engineering capabilities. Some questions to consider in capturing this in your strategy are:

- Can you produce a basic evaluation format for your software engineering environment, including tools, test beds, training, languages, design, metrics, and code development? What reasonable conclusions can you make?
- What are your greatest strengths in software engineering capability? What are your greatest weaknesses or risks?

- What controls or tracking methods are performed to recognize problems and mitigate risks?

- How are resources allocated for improvements in engineering functions?

- How stable are your processes for requirements collection, configuration management, and project control?

- How would you define the key factors that drive your designs and technical processes (e.g., the cost of customer recall, time to market, schedule, or cost)?

- What level of planning is used to ensure that software engineers have the appropriate technical training?

8. Customers

In forming your software strategy, it is essential to capture the role of the customer in relation to the software development organization. We find that firms often focus resources upon the software technology, processes, and sciences of development—without taking into consideration end users or customers. This is the failure of cultures that are highly technology intensive, and often leads to product or system functionality that fails to deliver the core values that customers require. Consider financial services firms, whose business is intangibles such as brokerage and banking services. We have coached customer-service teams to gather customer and user information about their monthly statements, and to use this market feedback in software strategy development. In other cases, we have worked with companies that have involved customers in product teams and co-development projects, to ensure that software development meets customers' growing or changing needs. This requires organizations to alter their internal software operations to accommodate greater involvement by customers. In short, great software often results where strategy is closely aligned to customers and end users. Some questions to consider in capturing this in your strategy are:

- How would you describe how your software organization keeps close to the customer? How do you keep abreast of customer preferences and values?

- How are your customers involved in ongoing software development efforts? Do you include customers and users in integrated product-development teams?

- To what degree does software create the product or service value that your customers demand?

- What is the greatest opportunity to increase customer satisfaction? How does this impact software?

- How would you briefly explain how technical improvements will deliver a better product or greater value to your customers?

- What is the most critical customer issue regarding your software (e.g., time, budget, quality, safety, function)?

- How would you describe the key problem of software from your customer's perspective?

Conclusion

Technology collaborations are essential to sustaining competitive advantages in software engineering development, but technology transition and collaboration planning require the support of good strategy formulation and planning.

While every company's software strategy should address those factors unique to its own business, software, and technical requirements, the above eight factors have repeatedly proven absolutely essential to software strategies that drive successful technology collaborations.

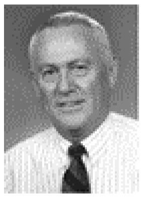
About the Author

Mike Mattison is a senior member of the technical staff at the Software Engineering Institute. Mike is currently managing a portfolio of technology collaborations with Fortune 500 client partners. The goal of his work is to accelerate the technical development and commercial transition of new software capabilities and methods into applied practice. He has directed a technology partnering strategy for SEI that has produced successful software collaborations with Fortune 300 corporations in the aerospace, automotive, banking, telecommunications, defense, and software industries. Mike is presently researching critical software performance in extreme high-reliability domains. His area of interest is the impact of software management performance on sustainable competitive advantage in commercial industries. You can reach Mike at mvm@sei.cmu.edu

The COTS Spot

Who’s in Charge Here?

David Carney



In my last column, I discussed the issue of requirements in the context of COTS-based systems, and described some of the changes that a commercial bias imposes on the way that systems are built (see “Requirements in COTS-Based Systems: A Thorny Question Indeed”). One of the major points of the article was that extensive use of COTS products brings with it an unavoidable loss of control over a system’s requirements. In this column I wish to explore that thesis a little more fully.

The Basic Idea

My essential premise is that we who are involved with government systems must realize, as we shift our posture in software acquisition toward a preference for commercial products, that we necessarily lose a significant amount of control over our software systems.

First off, this sounds dangerous, since a phrase like “losing control” has a certain ominous semantic ring. But before we hit the panic button, it’s important to recall that the government has willingly chosen to become a consumer; it should be no surprise that consumers generally have only partial control over the shape of the things they consume. To be sure, a consumer has some manner of control, in a very coarse-grained way, through his willingness to buy or to reject a given product. For instance, when Detroit puts out a car that few people like, marketplace rejection is certainly a form of control. But beyond that, it’s very doubtful that any given consumer can be said to *control* the type of motor that the automobile is designed to use, or to *control* the automobile’s shape, silhouette, or profile.

This is not altogether bad, since there are things that we really don’t want or need control over. Stretching the automobile analogy a little further, there are certainly some things that we do explicitly care about. We probably care what color the car is, and we often worry (some of us, anyway) about fuel efficiency. In addition, a buyer typically has some laundry list of other desirable features as well. But few of us car buyers really care too much about the torque (presuming we even know what torque is), and probably no one, not even the die-hard specialist, cares about the firing order of the pistons. For these things, we are just as happy that Ford or GM make all of the decisions, and we are perfectly happy to purchase products whose specifications are at least partially out of our control. (Even now I can hear the wails from the auto enthusiasts about torque…)

It’s All a Matter of Balance

Aside from whether we wish or don’t wish to make every decision about a system, it is nonetheless true that a loss of control is a necessary corollary to the benefits that come from a COTS bias. Switching to a somewhat different metaphor, the loss of control stemming from the use of COTS products is proof that some of the basic laws of physics hold equally for system acquisition. Let’s recall two essential laws of physics. For instance, Newton’s Third Law:

For every action there is an equal and opposite reaction.

And the Law of Conservation of Matter and Energy:

The total quantity of matter and energy available in the universe is a fixed amount and never any more or less (i.e., adding energy means reducing matter).

What is common to both of these physical laws is the notion of balance: if something happens on one side of the ledger then something also happens on the other side; nothing in physics is absolutely independent.

I claim that this balancing act shows up just as naturally in system acquisition as well. For instance, there are many significant benefits and savings that come from shifting to a COTS-based acquisition strategy, at least for certain kinds of software. It is immediately obvious that we gain in speed of deployment. Forgetting for the moment the unfortunate delay of bureaucratic overhead, then buying a commercial product means we get it immediately, not after several years’ development time. And it is no less beneficial that by using best-of-breed commercial software, we in the government have a fighting chance to avoid the kind of archaic and unmaintainable systems that still surround us on every side. By positioning our information systems to ride industry trends, we maintain the greatest potential for keeping our systems on an ongoing evolutionary path, and for keeping technological currency in this rapidly changing software world.

But these benefits, like those more tangible events that are subject to physical laws, do not occur in a vacuum. Like atomic fission, where the release of energy means a parallel reduction of matter, our gain (the newfound benefits mentioned above) is balanced by the cost; not just in terms of the purchase price, but also manifest in that we relinquish control over both our systems and the commercial components that comprise them. (Note: there’s no claim here that using COTS is some sort of “zero-sum game.” But there *is* a claim that when using COTS, you have to pay the piper.)

A Brief Tangent: Outsourcing

We need to be more precise in what this “loss of control” really implies. Since I like tangents, let me try wandering away for just a bit, to examine a term that is very often heard in the business world these days: outsourcing.

It is becoming common for an organization to take steps to avoid duplication of effort, particularly with those business functions that are not part of its core competency. Thus, an organization perceives that some ancillary product or service currently produced internally mirrors a product or service also available externally. Through outsourcing, the organization removes the unnecessary and costly duplication. For instance, imagine a large company whose employees travel frequently. That company might at one time have had an internal travel office to perform all travel-related services. Through outsourcing, the company makes use of an external travel agency to perform the same services. The cost of paying the external agency is more than offset by the savings in internal resources.

Note the key phrase here: *perform the same services*. The things done by the company’s internal travel bureau and the external travel bureau are essentially the same service. The internal service might have been a little quicker perhaps, or it may have done a little better in finding optimal flight times. But the basic services provided are the same, which is why the company is willing to outsource the service.

Now, it is sometimes heard (by this listener, at least) that the widespread move toward using COTS products is just a form of outsourcing. From my point of view, this is a misunderstanding of what it really means to use COTS (“to the maximum extent practicable,” as the mandate says). To be sure, there are certain parallels: both are partially motivated by a desire to cut cost, and both involve going outside an organizational boundary to procure something previously produced internally.

Probably the thing that most people mean when they claim that “using COTS is a form of outsourcing” is that they now purchase software rather than build it themselves in-house. They have “outsourced” the creation of their software infrastructure. The basis of this view is the realization that many—perhaps very many—government systems have few genuine differences from those used in the industrial world. For systems in the domains of payroll, accounting, inventory, and so forth, the reality is that most of the perceived government-specific requirements are illusory, and use of commercial products to support these services is both feasible and warranted.

But there are vital distinctions as well. With few exceptions, the products available in the COTS software marketplace only partially match with government business processes. By making the conscious choice to use commercial software technology, the government commits itself to whatever changed business processes are needed. The changes might be relatively painless (as in the earlier example of a business outsourcing a travel service), but it may well be far more substantial, requiring a governmental agency to make serious revision of its business processes. This becomes rather distant from the notion of "outsourcing." And, whatever the match between the COTS product and the government's business needs, using commercial products means that someone else controls the government's software infrastructure, which returns us to the main point.

The Effect of Losing Control

Assuming that we're willing to accept all this, what does it really mean in practice? In a nutshell, it would seem that the loss of control is especially obvious in requirements, in functionality, and in schedule.

We discussed the loss of control over requirements last month at length, so I will merely summarize here. My assertion was that requirements at one time sat alone in the driver's seat, and were foundational in determining how the system was built and how it worked. We all used, and rightly believed in, phrases like "get the requirements right." But in a COTS-based paradigm, this is less true: our requirements now have to share the driver's seat, and we may never get them quite right. To the extent that a given system incorporates commercial components, it incorporates pieces whose requirements were established independently of that system. Whether willingly or no, the builders of the system must permit its commercial components (and their implicit requirements) to have a say in the system's overall shape and functioning.

To be sure, "requirements" in the very widest sense still drive the creation of COTS software, since there is some set of technical objectives that govern how software—any software—is created. But those "requirements" are now the collection of features that a vendor believes will appeal to the widest set of potential customers; those requirements are the aggregate requirements of the whole marketplace, not of any one individual. This results in a kind of democracy of requirements: regardless of the feature you might care about, you're just one consumer among many, and quality is secondary to market share. One need go no further than everyday office software to see how profoundly this is true.

So the requirements that drive a commercial product, and hence, the functional workings of the product, are determined by its vendor. But that's not all; it doesn't stop there. The vendor is also the one who makes the ongoing decisions about which features stay and which are removed from future releases. The vendor makes the choices about long-term sustainment, and also makes the critical decision about whether the product continues to exist at all. The vendor has equal control over the product's release schedule, when upgrades are reissued, how often they occur, and what additional long-term license costs will appear. (This should not be a surprise: the vendor's goal is to make money. That's why he's in business, and *any* strategy we adopt that involves commercial software components should always assume and be based on this fundamental truth.

But What if We *Can't* Afford to Lose Control?

Supposing that we'll accept that the use of COTS leads to loss of control, whether of the things mentioned in the previous paragraph or any others. Isn't it also the case that there are some occasions when this condition is *not* acceptable? Aren't there circumstances when we really can't afford to lose control? And even more difficult: how can we recognize such circumstances?

Well, let's also go back to the Detroit analogy. Suppose we're not talking about automobiles, but about robotic vehicles that do something nifty with nuclear explosives. Do we seriously expect that the DoD should be willing to leave any decisions about how such a machine works (and especially its deep-down internals!) to a designer of commercial automobiles? Even more to the point: should we applaud if the DoD were to buy such vehicles as cheaply as possible?

It's fairly obvious that I've asked these questions rhetorically, and that I'm arguing against a COTS approach for such machines; they would probably not be good candidates for a COTS-based acquisition strategy. Conceivably, perhaps, some sub-components of such a vehicle might use commercial products, but even that kind of decision would be based on a firm sense that the system requirements are still dominant and not negotiable, that the engineering decisions are made by the robotic vehicles' designers, not by anyone else, and that the system is not being built to leverage market trends but to fulfill a very specific and life-critical mission.

As we said at the top, normal consumers don't control Detroit's plans, nor should they expect to. Most consumers tend to accept what the marketplace offers, and derive the benefits that come from marketplace competition. But some consumers aren't normal, either in schedule or in requirements or in functionality. Some people really do need to worry about torque.

Last Thoughts

We sometimes forget that the current drive toward using COTS is a means to an end, not the end itself. The real end is a complex mix of factors. Savings, sure; there's no room in the budget for the unnecessary duplication by the government of comparable capabilities in the industrial world. But there are lots of other factors as well. One is a realization by the government that it already has too many tasks, and overseeing large-scale production of business software is an unwanted burden. Even more, guidelines for the use of COTS should be based on an awareness that the lightning-fast technological revolution we're caught up in is independent of any single force, even a force as big as

Uncle Sam. He (and we) are far better off if we watch the trends as closely as possible, admit the reality of the marketplace, and hang on for dear life.

And hang on we must, given the frenzy of the commercial software marketplace. As Bette Davis, in the film *All About Eve*, predicted long ago: “Fasten your seat belts, everybody. It’s going to be a bumpy ride!”

Next time, some thoughts about COTS and risk management. Stay tuned.

Net Effects

The Net Effects of Product Lines

Scott Tilley

One of the most promising aspects of net-centric computing is its ability to aid the incremental migration from stove-piped legacy systems to product lines. Operational for many years, legacy systems are often viewed as liabilities, in part because of the difficulty involved in understanding their structures and upgrading their capabilities to meet new business requirements. Net-centric computing can play an important role in converting liabilities into assets through a three-step process of decomposing the legacy system into separate functional units, developing these artifacts into components using distributed object technology, and deploying these components as core assets in the product line.

As described elsewhere in this edition of *SEI Interactive*, a product line is a group of products sharing a common, managed set of features that together address a particular market segment or fulfill a particular mission. Product lines offer several advantages over more traditional application-development approaches, not the least of which is the ability to reuse corporate assets across a number of separate but similar products. However, in order to adopt a product-line approach to software engineering, the existing base of legacy systems must be dealt with first.

Legacy Systems

By definition, legacy systems are resistant to change. Although legacy systems vary in their individual features, they all share several characteristics. They are old (10–25 years) and large (100 KLOC–1 MLOC). Such age and size typically means the system is poorly structured, often due to traumatic maintenance over its lifetime. This contributes to a poor understanding of the system's essential functionality, a situation that is exacerbated by personnel turnover: each developer is forced to relearn aspects of the system that previous developers spent significant time and effort to recover. Nevertheless, legacy systems represent substantial corporate knowledge and cannot simply be discarded and rebuilt with a "green field" development effort.

Instead of viewing legacy systems as liabilities that are difficult to understand and hence difficult to evolve, a better approach is to turn them into valuable corporate assets. Product lines offer the ability to leverage existing assets to provide a significant return on investment over the long term. This can be accomplished by providing the infrastructure for the strategic reuse of corporate assets, such as mission-critical systems and associated business rules. This process is illustrated in the following figure.

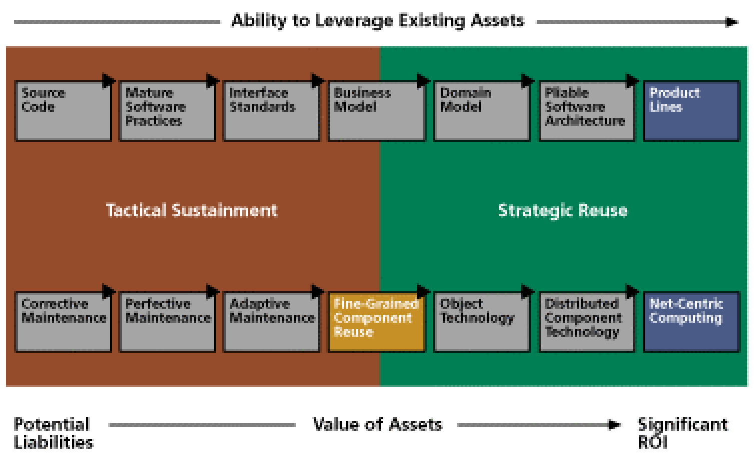


Figure 1: Leveraging Assets

The Migration Process

Migrating legacy systems to product lines can be accomplished using a three-step process. The first step is to decompose the legacy system into separate functional units. The second step is to develop these extracted artifacts into components using distributed object technology. The third step is to deploy the new components as core assets, making them available to the product line.

The result of the migration process is a three-tiered set of interacting components, as illustrated in Figure 2. Each component may reside on a client or a server, or it may migrate between them (in effect, serving multiple roles simultaneously). Data can also be made similarly mobile. These components can provide the basis for a product line as the software continues to evolve to meet changing business requirements.

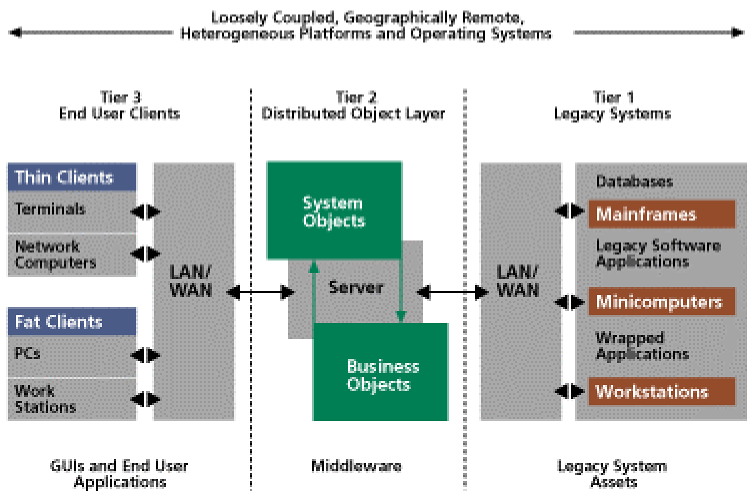


Figure 2: Three-Tiered Architecture

Decomposing the Legacy System

The first step in the migration process is to decompose the legacy system into separate functional units. This can be accomplished using two variants of system-understanding technology. The first variant is a white-box approach that relies on traditional program understanding. This is typically done using a reverse-engineering tool that parses the legacy system's source code, populates a repository with the data it gathers, and performs analysis on this data to aid the user in understanding the system.

The second variant is a black-box approach that uses newer methods to understand the system. These methods can include binary reverse engineering, interface analysis, and behavioral probing. The black-box approach is used when the source code to the legacy system is not available for white-box analysis.

The result of this procedure is the first tier of the new product-line infrastructure. At this point, the artifacts can be considered to be virtual components. They represent essential functional artifacts of the legacy system, but are not yet in a format suitable for use as a true component in the software engineering sense.

Developing the Core Assets

The second step in the migration process is to develop the product line's core assets from the artifacts extracted from the legacy system in the first step. This is done by wrapping the virtual components with object interfaces. This results in true components that become available for use in the third step of the migration process.

Wrapping the virtual components can be accomplished using distributed object technology as middleware. There are several different middleware offerings to choose from for this task. One of the most popular is the Object Management Group's Common Object Request Broker Architecture (CORBA). Microsoft is a proponent of the Distributed Component Object Model (DCOM) and related technologies under the COM+ rubric. Sun Microsystems espouses an approach relying on Enterprise Java Beans (EJB), possibly augmented with its Jini networked-appliance enabling technologies.

The result of this second step is the middle tier of the new product-line infrastructure. The new business artifacts are encapsulated with object interfaces, turning them into core assets and making them available to the third tier. The final step is to deploy the assets.

Deploying the Core Assets

The third step in the migration process is to deploy the core assets using the network infrastructure. The core assets are accessed by product-line developers, who in turn use the assets to construct variant product instances. The result of this third step is the final tier of the new product-line infrastructure.

By deploying the core assets across a network, client applications can access and browse the collection of core assets. This facilitates the strategic reuse of business-critical core assets, which has two benefits. The first benefit is the ability to leverage existing capabilities—in other words, to convert legacy liabilities into valuable product-line assets.

The second benefit is the ability to compose new functionality from the newly mined business objects. Because the virtual components that were mined from the legacy system in the first step of the process have been given object-oriented interfaces in the second step of the process, the resultant artifacts can be combined to provide functionality that would have required significant writing of new code in a non-product-line approach. This composition of assets into new functionality facilitates the evolution of the legacy system to meet new requirements in a disciplined and cost-effective manner.

The Net Effects of Product Lines

Today's newly developed system is tomorrow's legacy system. As such, legacy systems will always be with us, and will remain vitally important for the foreseeable future. The key to successfully adopting a product-line approach to software engineering is converting existing legacy systems from liabilities into assets. This can be accomplished using the three-step process described above.

In essence, net-centric computing provides the same application and asset portability to the client that distributed object technology provides to the server. The result is a flexible deployment infrastructure for core assets of the product line. The three-tiered architecture can be used by both end-users of the derived products, and by developers who rely on the core assets to construct instances of the product line.

The net effects of product lines are that they make software engineering more akin to a manufacturing process, with predictable attributes such as cost, schedule, and quality. The key is the strategic reuse of corporate assets—assets that are usually considered liabilities. Product lines promise to pervade software engineering in the new millennium, resulting in the development of better products in shorter time at lower cost. Underlying the success of product lines is the infrastructure provided by net-centric computing for migrating legacy systems from stove-piped systems to deployed core assets.

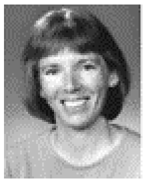
About the author

Scott Tilley is a visiting scientist with the Software Engineering Institute at Carnegie Mellon University, an assistant professor in the Department of Computer Science at the University of California, Riverside, and principal of S.R. Tilley & Associates, an information technology consulting boutique. He can be reached at stilley@sei.cmu.edu.

Security Matters

From Y2K to Security Improvement: A Critical Transition

Moira West-Brown, Julia Allen



The previous issue in this series discussed how the Internet community could better prepare to address major security incidents. In this issue I'm joined by Julia Allen, team leader for security-improvement practice development. We will compare Y2K and information technology (IT) security and suggest how your organization can build on its Y2K efforts to initiate or enhance an IT security-improvement program (SIP).

The Basis for Security Improvement

Many organizations have spent the past year preparing for the impact of Y2K. Unfortunately few of them are ready to address the risks associated with IT security incidents. The actions that must be taken to successfully deal with such incidents need to be a continuous, planned part of normal, day-to-day business operations. As in preparing for Y2K, IT security needs visible management sponsorship, investment, policies, procedures, processes, methods, tools, measures, standing teams, and assigned roles and responsibilities. This combination of people, technology, and processes is the basis for security improvement.

Y2K vs. IT Security—A Comparison

As the Y2K deadline looms ever closer, organizations find themselves in various stages of readiness for the big event. After years of work, some are still frantically attempting to complete their Y2K compliance testing, while others are preparing their crisis communications to monitor for and address Y2K failures and glitches when they do arise. However, as the new millennium dawns, organizations can expect to address more than just Y2K glitches and failures.

The tremendous energy that organizations have exerted to prepare for Y2K is understandable when you consider what is at stake. However, the risks associated with suffering an IT security incident or security breach can be just as devastating as those associated with Y2K non-compliance. Recent figures provided in the 1999 CSI/FBI computer crime survey indicate that the greatest losses from IT security incidents are associated with theft of proprietary information and financial fraud. But the survey also points out that many organizations are unable to quantify losses from incidents. All too often organizations either don't know what information may have been lost or don't have processes in place to help determine how to quantify loss. For instance, what is the cost to an organization of losing its Internet connectivity for five hours?

Moreover, organizations may naively believe that IT security is under control if they have some security measures in place—such as

- a firewall to keep out intruders
- investment in PKI (public key infrastructure), VPN (virtual private network), or e-commerce solutions
- an IDS (Intrusion Detection System) to detect, alert, and possibly respond to intrusions
- strong authentication using technologies such as one-time passwords and smart cards

It is important to recognize that mitigating IT security risks is a complex issue that can neither be addressed overnight nor through technological solutions alone.

A survey by the SANS Institute of 1,850 computer-security experts and managers identified "Seven Top Management Errors that Lead to Computer Security Vulnerabilities."

For some time, computer-security experts have warned of the possibility of intruders using the chaos and confusion of Y2K as a smokescreen under which they can camouflage attacks and other malicious activities. Recently the Gartner Group has asserted the potential for someone to steal up to $1 billion during the Y2K chaos by installing back doors in software during Y2K compliance changes.

Unlike Y2K—fixing a one-off issue at a known time in the future—IT security incidents are a reality now, occur on a daily basis, and may prove at least as catastrophic for a company as Y2K. Keeping pace with changing business and technology demands results in dynamic IT environments with ongoing changes to platforms, tools, technologies, staff and policies. Keeping pace is difficult enough from a Y2K perspective; an IT-security perspective adds additional levels of complexity to the problem and correspondingly increases risk.

Although the risks of Y2K and IT security to an organization are comparable, recognition of this is not reflected in the associated level of investment needed to address these risks. No one knows the real global cost of Y2K, but figures available in the U.S. give an indication of the magnitude of the investment that companies are making to address this one-off event. Organizations have invested and continue to invest significant sums in their Y2K compliance efforts. In July 1999, for example, U.S. and Canadian airlines reported that their combined Y2K efforts totaled more than $750 million. In December 1999, the total reported Y2K costs for U.S. federal agencies will reach over $6.4 billion. Even considering the enormous sums reported, some experts claim that organizations are underreporting their real Y2K costs so as not to reduce customer confidence.

Figures on investment in IT security improvement are more difficult to obtain, but the little information that is available would indicate that global IT security investment is embarrassingly small in comparison with Y2K budgets. The Gartner Group estimates that most organizations spend as little as 1% of their operating costs on security when 5–8% is what is necessary. In a 1999 survey, *Information Week* showed that approximately 50% of information security professionals had an IT security budget of $50,000 or less.

Y2K and security improvement are corporate-wide issues that could have serious repercussions if not adequately addressed. Just as with Y2K, launching and sustaining a successful security-improvement program requires visible advocacy by senior management, funding, follow-through, and long-term commitment of resources.

Initiating a Security Improvement Program

Initiating a security improvement program (SIP) is hard work, even if you’ve had a significant attack that has gotten everyone’s attention. Sustaining an SIP can be even harder. First, you need to identify the risks to your business if the security (confidentiality, availability, and integrity) of critical data, systems, and/or networks (assets) is compromised. By compromised, we mean that the asset has been destroyed, damaged, altered so as to hurt your operations, or revealed to your competitors. You can’t protect everything equally so it is important to carefully select what you do choose to protect and how, based on its value to your organization.

Once you know your risks, you need to decide which ones are most likely to occur and have the largest potential impact. Impact could be in dollars, time, lost productivity, or loss of market share, customers, and reputation. But the work doesn’t end there. Let’s say you have a prioritized list of risks and an effective plan to mitigate them. The next day, you go into the office and find out your number one competitor has just launched a new e-commerce site and is ready to do business on the Internet—and you’re still six months away from launching yours. Or a recently fired employee has successfully penetrated your strategic planning database and posted your plans for the next 18 months on an Internet news group. In other words, change and surprises introduce new risks that must be added to the ones you are already managing. And you need to have a way of adjusting where you invest SIP time and energy based on this very dynamic environment.

In concert with the CERT/CC® community and several leading government and commercial organizations, we have spent some time thinking about how to launch an SIP[2]. One of the key components of an SIP is the definition and adoption of improved security practices that will allow you to mitigate your most critical technical risks.

When considering who could most benefit from pragmatic, concise, how-to guidance on what to do (practices), it became obvious that one of the audiences with the greatest need was network and system administrators and their managers. They face the most daunting challenges as a result of the growth and complexity of the IT infrastructures they are responsible for keeping up and running 24 hours a day, seven days a week. And they are constantly being asked to add new IT systems, networks, applications, and data to keep pace with changing business and technology demands. Based on what successful organizations were doing to deal with these demands, we developed specific step-by-step guidance that did not rely on a particular operating system or platform, making the information as broadly useful as possible. In addition, UNIX- and Windows NT-specific "implementations" for many of the practices have been developed. All of this information can be found on the CERT security improvement Web site.

Each practice contains

- a brief description that expands the title of the practice
- an explanation of why the practice is important (what bad things can happen if you don't do it)
- a step-by-step description of how to perform the practice
- related policy topics that support successfully deploying the practice

Planned future additions include

- cost/benefit analysis information for selecting among alternative approaches
- the means by which to measure success of implementation (did it solve the problem it purported to solve and were the benefits of the investment worth the cost)

[2] See our 1999 SEPG presentation on Securing Networked Systems

Some of the more frequently referenced sets of practices (each set is called a module) include Preparing to Detect Signs of Intrusion, Detecting Signs of Intrusion, Responding to Intrusions; Securing Desktop Workstations, Securing Network Servers, and Deploying Firewalls. The modules contain practices such as identifying and installing tools, setting up logging options and examining what they produce, setting up user authentication and file access control mechanisms, and determining how to deny network traffic that you don't want coming into your system.

(Many of the practices are starting to appear in training materials and are being referenced by other Web sites. We don't have any feedback yet on how organizations are using them but we would love to hear from you if you are. We are launching our first significant set of pilot tests this year. So stay tuned and watch for new materials.)

Transitioning Y2K Resources

As Y2K efforts wind down and resources associated with them free up, many projects that have been on hold or have been placed on the back burner will be competing for those resources. It is important to plan for the future now and ensure that IT security improvement is a major focus of those plans. In addition to redirecting Y2K resources to other development projects, this is an excellent opportunity to transition some of those resources to the formation of an SIP effort and to supplement these resources with IT-security expertise. This approach is preferable to resourcing SIP from scratch, as the people coordinating Y2K efforts in organizations are likely to be familiar with many of the issues that you need to address for SIP including

- establishing a crisis center
- a good understanding of the nature and level of risk across the organization
- identifying critical resources
- establishing contacts across business units

Some organizations are already considering this approach[3].

Security improvement won't happen overnight; it will result from an ongoing effort. Organizations need to be prepared to address IT-security incidents every day. However, every organization should also consider the heightened possibility of security incidents

[3] See recent issues of *Federal Computer Week* articles (Y2K and the CIO Council).

coinciding with Y2K. We encourage you to alert your Y2K crisis center to be prepared for possible security problems disguised as Y2K issues or anomalies that may coincide with apparent Y2K problems. Have IT security staff on alert to address any such issues as they arise.

Many organizations scrambled to address the Melissa macro virus incident earlier this year. Some have indicated that they were in some way thankful for the experience Melissa gave them as they were then better prepared when the potentially more severe explore.zip worm struck just months later. Organizations should take little solace from such news—clearly this is not an effective or appropriate way to address security risks. Y2K has taught us that having a deadline to shoot for can help us to focus our attention and make significant progress toward addressing a major problem. We don’t deny that security improvement is a much more complex and difficult nut to crack than Y2K. However, to retain control of corporate assets and continue to enhance the nature of the business conducted on the network while maintaining customer confidence, we must tackle security improvement head-on.

About the Authors

Moira J. West-Brown is a senior member of the technical staff within the CERT® Coordination Center, based at the SEI, where she leads a group responsible for facilitating and assisting the formation of new computer security incident response teams (CSIRTs) around the globe. Before coming to the CERT/CC in 1991, West-Brown had extensive experience in system administration, software development, and user support/liaison, gained at a variety of companies ranging from academic institutions and industrial software consultancies to government-funded research programs. She is an active figure in the international CSIRT community and has developed a variety of tutorial and workshop materials focusing mainly on operational and collaborative CSIRT issues. She was elected to the Forum of Incident Response and Security Teams Steering Committee in 1995 and is currently the Steering Committee Chair. She holds a first-class bachelor’s degree in computational science from the University of Hull, UK.

Julia Allen has more than 25 years of managerial and technical experience in software engineering. She is currently a senior member of the technical staff within the Networked Systems Survivability Program at the Software Engineering Institute, leading the team responsible for developing security improvement practices. Prior to this technical assignment, Allen served as acting director of the SEI for an interim period of six months as well as deputy director for three years. She started the Industry Customer Sector at the SEI in 1992. Before joining the SEI, Allen was vice president at Science Applications International Corp., and was responsible for starting a new software division specializing in embedded systems software for government customers. Before that, she worked for 10 years with TRW. Allen received a BS in Computer Science from the University of Michigan, as well as an MS from the University of Southern California and an executive business certificate from the University of California at Los Angeles. Her professional affiliations include ACM, IEEE Computer Society, and the Internet Society (ISOC). Her publications include four modules within the SEI's security improvement series as well as various presentations and papers on the SEI's strategic plan and technical program.

Watts New?

Getting Management Support for Process Improvement

Watts S. Humphrey

Over the years, I have often been asked about how to get management support for process improvement. Typically, engineers want to use better software methods but they have found that their management either doesn't care about the methods they use or, worse yet, even discourages them from trying to improve the way they work. In addressing this subject, I have decided to break it into two parts. The first part, which I covered in the June 1999 issue of *SEI Interactive*, concerns disciplined work: what it is, and what it takes to do it. In this column I address the problem of getting management support for process improvement.

Obtaining Broad Management Support

Perhaps the biggest problem in starting an improvement effort is getting management support. The first and most important step is to get senior management backing. Without support from the very top, it is generally impossible to make significant changes. Next, however, you will need active involvement from all the appropriate managers, particularly those managers who directly supervise the work to be impacted by the change.

The reason for broad management support is that significant improvement programs generally involve substantial changes in the way people work. If you don't change the engineers' working practices, you can change the organizational structure and all its procedures, but nothing much will really change. Thus, to have a substantial impact on an organization's performance, you must change the way the engineers actually work. While this is possible, it is very difficult, and it requires the support of all levels of management. Senior managers must establish goals and adjust reward systems. Intermediate managers need to provide funding and change priorities. And most important, the working-level managers must make the engineers available for training, support process development, and monitor the engineers' work to make sure they follow the improved practices. So, how do you get this kind of support? To address this question, we discuss three issues:

1. Why do you want to make changes?
2. Which managers do you need support from?
3. Why should those managers support you?

Why do you want to make changes?

Since you are reading this column, you are probably interested in making process changes, and these changes are undoubtedly in the way your organization develops or maintains software. This means you are probably talking about some kind of process improvement, like getting a Capability Maturity Model® (CMM)® program underway or introducing the Personal Software Process^{SM} (PSP^{SM}) and Team Software Process^{SM} (TSP^{SM}). Whatever the approach, you will be changing the way software work is done.

The first question to address is: why? That is, why do you want to improve the software process, why should management support you in improving the software process, and why should the organization care about how software is developed? These are tough questions, but they are the very first questions managers will ask. You need to be able to answer these questions, and depending on which managers you talk to, they will ask these questions differently. This leads us to the next question.

Which managers do you need support from?

Depending on the size of your organization, there could be many management levels. Typically, the manager from whom most of us need support is the manager immediately above us. While there are lots of levels to discuss, let me assume that this immediate manager runs a project or a department. Unless you are in a very small organization, this manager probably works for some higher-level manager, and this higher-level manager probably works for some manager at an even higher level. Up there somewhere there should be a senior-level manager or executive who is concerned with the overall business, how it performs now, and how it will perform in the future. This senior manager is concerned with where the business stands competitively, how new technology will impact its products and services, and the changing needs of its customers.

The reason the manager's level is important to you is that improvement programs focus on long-term issues that are the principal concern of senior-level executives. Unless the managers below the executive level are specifically charged with working on process improvement, most of them will view improvement efforts as a distraction at best or, at worst, as a drain on critical resources.

The reason for this negative view is that process improvement deals with the overall performance of an organization. It concerns competitive capabilities, long-term cost effectiveness, development cycle-time improvement, and customer satisfaction. These are strategic issues that generally only concern the most senior executives. Even in the departments, laboratories, or divisions of large corporations, the performance measures for division general managers, laboratory directors, and department managers are invariably concerned with immediate short-term results: delivering products on time, managing tight budgets, or responding to customer-related crises.

While these issues are critically important, and they often spell the difference between organizational failure and success, a total concentration on these topics will not change the way organizations perform. If the organization is not cost competitive, or if it produces lower quality or less attractive products, a focus on current performance will not improve the situation. The immediate problems may be fixed and the burning issues resolved, but the organization will continue working pretty much as it always has. It will thus continue producing essentially the same results and generating essentially the same problems and issues. This brings us to the definition of insanity: doing the same thing over and over and expecting a different result.

Generally, only the managers who think strategically will support a process-improvement program. These are usually managers who have broad business responsibilities and are measured by total organizational performance. They probably have multiple functions reporting to them, like product development, marketing, manufacturing, and service.

Even senior managers, however, do not always think strategically. Most organizations, after all, are owned by stockholders who are interested in the stock price. And since the stock price is heavily influenced by quarterly financial results, even the most senior managers cannot afford to ignore short-term financial performance. Unfortunately, many of these managers don’t worry about much else.

Why should this manager support you?

Now we get to the critical question: Why should any manager support you? In general terms, there are three reasons why managers might be willing to support you:

1. What you want to do supports their current job objectives.
2. What you want to do will make them look good to their immediate and higher-level managers.
3. What you want to do is so clearly right that they are willing to support you in spite of its impact on their immediate performance measures.

Getting help from a senior manager

The relative importance of these reasons changes, depending on where the manager resides in the management chain. At the very top are the managers who are most likely to focus on long-term performance. This means that they will often support process improvement for all three reasons. Thus, if you can show that process improvement will have a significant long-term benefit, you will likely get support. You can generally accomplish this by showing how similar improvements have benefited other organizations or, better yet, how they have benefited other parts of your own organization.

For the CMM, for example, show how improvements in CMM level have improved the performance of other software organizations. Also, show where your organization stands compared with other organizations in your industry. For the PSP and TSP, you could show data on quality, productivity, or employee turnover and how such changes could impact your organization.

If you can get the attention of a senior manager, and if you have your facts straight, the odds are you can get this manager to seriously consider the subject of process improvement. Frequently this is when you might get an outside expert to give a talk or to do an assessment. While you may have to settle for a small initial step, the key is to get some action taken. Once you can get the ball rolling, it is usually easier to keep it in motion.

If the manager you are dealing with is not at the senior executive level but one level lower, this manager is probably not measured on strategic issues. Such managers would know, however, that their immediate manager had such a measure. Thus, your manager is not likely to be motivated by reason 1 but might be persuaded to support you for reason 2. Thus, by proposing something that will make him or her look good to higher-level managers, this manager will personally benefit while also helping you to get the improvement ball rolling. What you want to ask for from this manager is help in taking the improvement story upstairs.

Getting help at the first management level

Finally, the most common problem is dealing with a manager who is fairly far down in the organization. This manager not only is not measured on strategic issues, but his or her immediate manager is not either. This means that strategic objectives are not likely to be very compelling. At this point, you only have two choices:

- Convince this manager that the improvement is a strategic necessity for the organization.

- Show how the improvement effort can help to address immediate short-term concerns.

While the latter is often the approach you must take, it has a built-in trap. The reason is that if improvement is aimed at solving a short-term problem, as soon as the short-term pain is relieved, the need for improvement is gone. This is like taking aspirin for a splitting headache. If the headache is indeed a transient problem, that would be appropriate. If the pain is the first symptom of a stroke or a brain tumor, however, the delay could be fatal. While promptly taking an aspirin may usually be helpful for a stroke, you had better also see a doctor right away.

In the software process, the problems in most organizations are more like strokes and brain tumors than they are like headaches. While you may have no choice but to sell the improvement effort as a short-term solution, try to move to strategic issues as soon as you can get the attention of someone upstairs.

The next questions concern making the strategic case for improvements, making the tactical case, and moving from a tactically based to a strategically based improvement program. These will be topics of future columns.

Acknowledgements

In writing papers and columns, I make a practice of asking associates to review early drafts. For this column, I particularly appreciate the helpful comments and suggestions of Dan Burton, Jean-Marc Heneman, Julia Mullaney, and Bill Peterson.

An Invitation to Readers

In these columns, I discuss software issues and the impact of quality and process on engineers and their organizations. I am, however, most interested in addressing issues you feel are important. So please drop me a note with your comments, questions, or suggestions. I will read your notes and consider them when I plan future columns.

Thanks for your attention and please stay tuned in.

About the Author

Watts S. Humphrey founded the Software Process Program at the SEI. He is a fellow of the institute and is a research scientist on its staff. From 1959 to 1986, he was associated with IBM Corporation, where he was director of programming quality and process. His publications include many technical papers and six books. His most recent books are: *Managing the Software Process* (1989), *A Discipline for Software Engineering* (1995), *Managing Technical People* (1996), and *Introduction to the Personal Software Process*^{SM}(1997). He holds five U.S. patents. He is a member of the Association for Computing Machinery, a fellow of the Institute for Electrical and Electronics Engineers, and a past member of the Malcolm Baldridge National Quality Award Board of Examiners. He holds a BS in physics from the University of Chicago, an MS in physics from the Illinois Institute of Technology, and an MBA from the University of Chicago.