Der nachstehende Text ist ein Zusammenzug der Webseite:

https://www.sei.cmu.edu/productlines/frame_report/introduction.htm

Der Firma **Software Engeneering Institute**

Other Ways to Contact Us

You can reach us by phone, email, fax, or postal mail.

Phone: 412-268-5800
Toll Free: 1-888-201-4479

Email: info@sei.cmu.edu

Fax: 412-268-5758

Postal Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Business hours: Monday - Friday, 8:30 a.m. - 4:30 p.m. EST

Benefits and Costs of a Product Line

Software product line approaches accrue benefits at multiple levels. This section lists the benefits (and some of the costs) from the perspective of the organization as a whole, the individuals within the organization, and the core assets involved in software product line production.

Organizational Benefits

The organizations that we have studied[1] have achieved remarkable benefits that are aligned with commonly held business goals including

- large-scale productivity gains
- decreased time to market
- increased product quality
- decreased product risk
- increased market agility
- increased customer satisfaction
- more efficient use of human resources
- ability to effect mass customization
- ability to maintain market presence
- ability to sustain unprecedented growth

These benefits give organizations a competitive advantage and are derived from the reuse of the core assets in a strategic and prescribed way. Once the core asset base for the product line is established, there is a direct savings *each* time a product is built. That savings is associated with *each* of the following:

- **requirements:** There are common product line requirements. Product requirements are deltas to this established requirements base. Extensive requirements analysis is saved, and feasibility is assured.
- **architecture:** An architecture for a software system represents a large investment of time from the organization's most talented engineers. The quality goals for a system–its performance, reliability, modifiability, and so on–are largely allowed or precluded once the architecture is in place. If the architecture is wrong, the system cannot be saved. The product line architecture is used for each product and need only be instantiated. Considerable time and risk are spared.
- **components:** Up to 100% of the components in the core asset base are used in each product. These components may need to be altered using inheritance or parameters, but the design is intact, as are data structures and algorithms. In addition, the product line architecture provides component specifications for all the non-unique components that may be necessary.
- **modeling and analysis:** Performance models and the associated analyses are existing product line core assets. With each new product, there is extremely high confidence that the timing problems have been worked out and the bugs associated with distributed computing–synchronization, network loading, and absence of deadlock–have been eliminated.
- **testing:** Generic test plans, test processes, test cases, test data, test harnesses, and the communication paths required to report and fix problems have already been built. These testing artifacts need only be tailored on the basis of the variations related to the product.
- **planning:** The production plan has already been established. Baseline budgets and schedules from previous product development projects already exist and provide a reliable basis for the product work plans.
- **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:** Fewer people are required to build products, and the people are more easily transferred across the entire line.

A software product line approach provides options to future market opportunities. Even though exact opportunities and their certainty are impossible to predict, by exercising variation points in the core assets, product lines permit low-cost, low-risk experiments to explore opportunities.

Product lines enhance quality. Each new system takes advantage of all the defect elimination in its forebears; developer and customer confidence both 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 once for the entire family.

Individual Benefits

The benefits to individuals within an organization depend on their respective roles. The following table shows observed benefits for some of the individual stakeholders in the product line organization.

Product Line Benefits for Individual Stakeholders	
Stakeholder Role	**Benefits**
Chief executive officer (CEO)	Options to quickly develop new products; large productivity gains; greatly improved time to market; sustained growth and market presence; the options and ability to economically capture a market niche
Chief operating officer (COO)	Efficient use of workforce; ability to explore new markets, new technology, and/or new products; fluid personnel pool
Technical manager	Increased predictability; well-established roles and responsibilities; efficient production
Software product developer	Higher morale; greater job satisfaction; can focus on truly unique aspects of products; easier software integration; fewer schedule delays; greater mobility within the organization; more marketable; have time to learn new technology; are part of a team building products with an established quality record and reputation
Architect or core asset developer	Greater challenge; work has more impact; prestige within the organization; become as marketable as the product line
Marketer	Predictable high-quality products; predictable delivery; can sell products with a pedigree
Customer	High-quality products; predictable delivery date; predictable cost; known costs for unique requirements; well-tested training materials and documentation; shared maintenance costs; potential to participate in a user's group
End user	Fewer defects; better training materials and documentation; a network of other users

Benefits Versus Costs

We have established that the strategic reuse of core assets that defines product line practice represents an opportunity for benefits across the board, but the picture is not yet complete. Launching a software product line is a business decision that should not be made randomly. Any organization that launches a product line should have in mind specific and solid business goals that it plans to achieve through product line practice. Moreover, the benefits given above should align carefully with the achievement of those goals, because a software product line requires a start-up investment as well as ongoing costs to maintain the core assets. We have already listed the benefits associated with the reuse of particular core assets. Usually a cost and a caveat are associated with the achievement of each benefit. The following table gives a partial list of core assets with the typical additional costs. We repeat the benefits for the sake of comparison.

Costs and Benefits of Product Lines		
Core Asset	**Benefit**	**Additional Costs**
Requirements: The requirements are written for the group of systems as a whole, with requirements for individual systems specified by a delta or an increment to the generic set.	Commonality and variation are documented explicitly, which will help lead to an architecture for the product line. New systems in the product line will be much simpler to specify, because the requirements are reused and tailored.	Capturing requirements for a group of systems may require sophisticated analysis and intense negotiation to agree on both common requirements and variation points that are acceptable for all the systems.
Architecture: The architecture for the product line is the blueprint for how each product is assembled from the components in the core asset base.	Architecture represents a significant 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 largely completed.	The architecture must support the variation inherent in the product line, which imposes an additional constraint on the architecture and requires greater talent to define.
Software components: The software components that populate the core asset base form the building blocks for each product in the product line. Some will be reused without alteration. Others will be tailored according to prespecified variation mechanisms.	The interfaces for components are reused. For actual components that are reused, the design decisions, data structures, algorithms, documentation, reviews, code, and debugging effort can all be leveraged across multiple products in the product line.	The components must be designed to be robust and extensible so that they are applicable across a range of product contexts. Variation points must be built in or at least anticipated. Often, components must be designed to be more general without loss of performance.
Performance modeling and analysis: For products that must meet real-time constraints (and some that have soft 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, creating new processes, or synchronizing existing processes.
Business case, market analysis, marketing collateral, and cost and schedule estimates: These are the up-front business necessities involved in any product. Generic versions	All the business and management artifacts involved in turning out the product line already exist at least in a generic form and can be reused.	All these artifacts must be generic or be made extensible to accommodate product variations.

are built that support the entire product line.		
Tools and processes for software development and making changes: The infrastructure for turning out a software product requires specific product line processes and appropriate tool support.	Configuration control boards, configuration management tools and procedures, management processes, and the overall software development process are all in place and have been used before. Tools and environments purchased for one product can be amortized across the entire product line.	The boards, process definitions, tools, and procedures must be more robust to account for unique product line needs and the differences between managing a product line and managing a single product.
Test cases, test plans, and test data: There are generic testing artifacts for the entire set of products in the product line with variation points to accommodate product variation.	Test plans, test cases, test scripts, and test data have already been developed and reviewed for the components that are reused. Testing artifacts represent a substantial organizational investment. Any saving in this area is a benefit.	All the testing artifacts must be more robust, because they will support more than one product. In addition, they must be extensible to accommodate variation among the products.
People, skills, and training: In a product line organization, even though members of the development staff may work on a single product at a time, they are, in reality, working on the entire product line. The product line is a single entity that embraces multiple products.	Because of the commonality of the products and the production process, personnel can be more easily transferred among product projects as required. Staff expertise is usually applicable across the entire product line. Their productivity, when measured by the number of products to which their work applies, rises dramatically. Resources spent on training developers to use processes, tools, and system components are expended only once.	Personnel must be trained beyond general software engineering and corporate procedures to ensure that they understand software product line practices and can use the core 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 toward relevant domain expertise and technology forecasting. This transition must be managed.

For each of these core assets, the investment cost is usually much less than the value of the benefit. Also, most of the costs are up-front costs associated with establishing the product line. The benefits, on the other hand, accrue with each new product release. Once the approach is established, the organization's productivity accelerates rapidly, and the benefits far outweigh the costs. 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.

It takes a certain degree of maturity in the developing organization to field a product line successfully. Technology change is not the only barrier to successful product line adoption. Changes in management and organizational practices are also involved. Successful adoption of software product line practice is a careful blend of technological, process, organizational, and business improvements.

Organizations of all stripes have enjoyed quantitative benefits from their product lines. Product line practitioners have also shared with us examples of the costs, such as

- canceling three large projects so that sufficient resources could be devoted to the up-front development of core assets
- reassigning staff who could not adjust to the product line way of doing business
- suspending product delivery for an extended period of time while putting the new practices into place

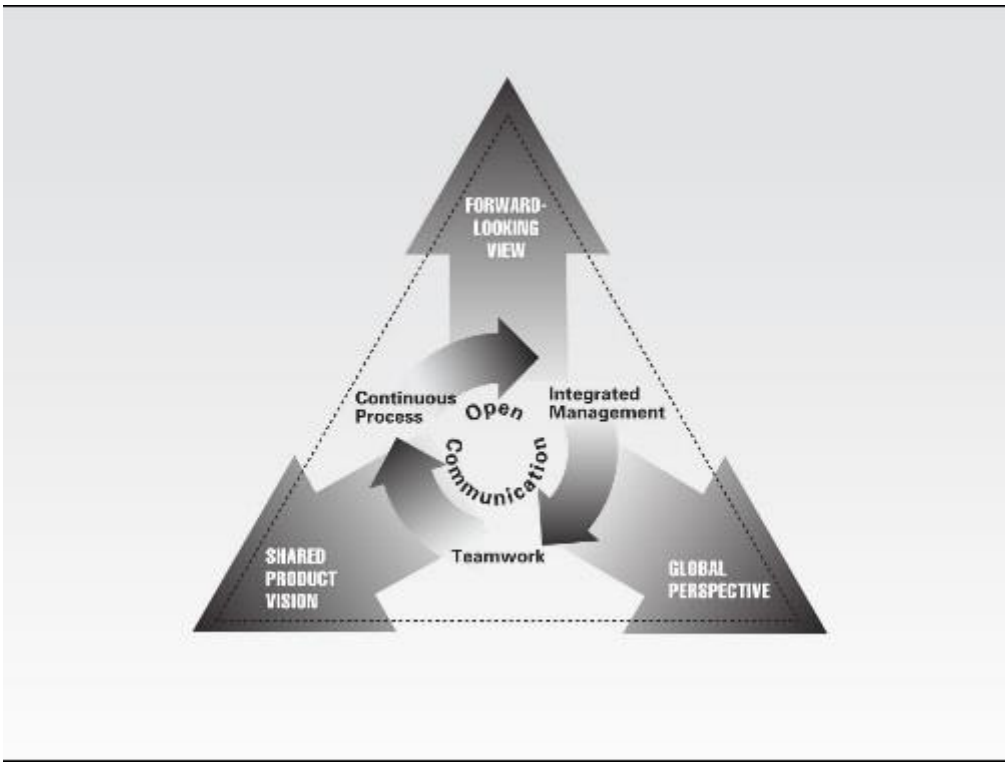
Certainly not every organization must undergo such dramatic measures to adopt the software product line approach. And the companies that bore these costs and made the successful transition to product line practice all agree that the payoff more than compensated for the effort. However, these costs underscore the point that product line practice is often uncharted territory and may not be the right path for every organization.

A Framework for Software Product Line Practice, Version 5.0

Organizational Risk Management

Organizational risk management is risk management at the strategic level. Risk management concepts are discussed in the "Technical Risk Management" practice area, which describes the activities that are necessary for project-level risk management. An organizational risk management process relies on the existence of such project-level risk management and provides mechanisms for surfacing and managing risks that transcend, or are shared across, projects.

The seven principles of risk management are shown in the following figure. These principles are divided into one core principle, three sustaining principles, and three defining principles. An effective risk management program exhibits characteristics of all seven principles.



The Seven Principles of Risk Management

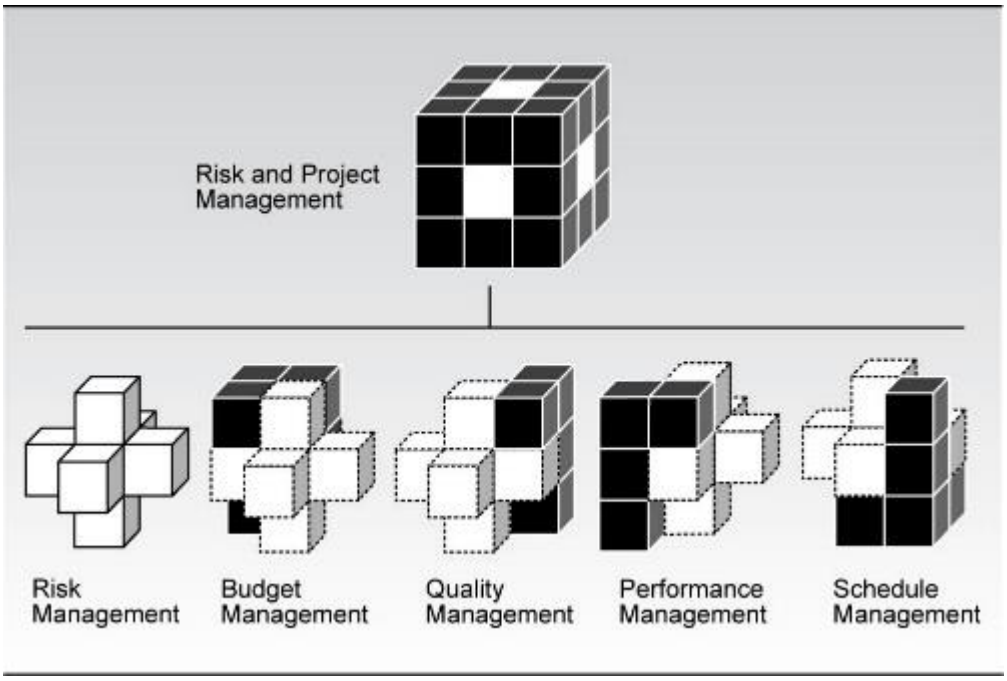
Core Principle

- **open communications:** An effective risk management process must encourage the free flow of information at and between all project levels. The process should enable formal, informal, and impromptu communication.

Sustaining Principles

Three sustaining principles allow an active risk management process to sustain its success in an organization:

- **integrated management:** When risk management is treated as a "bolt-on" or side activity, team members become frustrated, because managing risk is seen as interfering with their "real work." Teams performing risk management must integrate risk identification, risk analysis, and risk planning tracking and control into normal product line management activities and forums. As depicted in the following figure, risk management is integral to project management.



Risk and Project Management

- **teamwork:** The success of any software development project hinges on good teamwork. Team Risk Management (see figure below) brings together groups with diverse and sometimes conflicting objectives and allows them to discuss risks that cross project boundaries, figure out which risks are most important to mitigate, and pool resources to lower the risk exposure facing all parties.
- continuous process: The risks to an organization change constantly. The key to instituting a continuous process is to maintain the surveillance of risks as they change and to constantly identify new risks. Warren Keuffel warns against complacency:

  A software development project's environment is constantly changing, as a result of competitive pressures, organizational strategy and personnel changes, and technical challenges. Too frequently, a risk management plan, like a system architecture design, is prepared at the beginning of the project and then shelved. To be valuable, risk management needs to be revisited as frequently as schedule and technical issues [Keuffel 1999a].

Defining Principles

Three defining principles help to ensure success when working in a complex, multi-project environment:

- **forward-looking view:** Everyone is focused on the same tomorrow. Inevitably, when dealing with multiple, interrelated projects that focus on the same success point, conflicts will arise. The organizational management will need to keep all parties focused on the same success point.
- **global perspective:** Everyone has the same definition of success. From an organizational point of view, it may mean lower operations and maintenance costs over a 20-year period. It's organizational management's responsibility to articulate the business goals clearly and define success from a global perspective.

- **shared product vision:** Everyone sees the final capability as the same thing. The entire group of related developers, technical managers, and organizational managers must have a clear vision of where the organization is headed.

Aspects Peculiar to Product Lines

Product line efforts require a great deal of coordination across project boundaries, which makes well-defined, organizational risk management practices essential. As the organization orchestrates the product line effort, it identifies risks that affect multiple core asset developers and/or product developers.

The core asset and product development teams will manage the risks to their individual projects. (For more information, see the "Technical Risk Management" practice area.) Each project management team will have cost, schedule, and technical objectives that are peculiar to the task they are trying to accomplish. When risks cross multiple projects or affect the organization's successful implementation of a product line approach, the risks should be managed at the organization level.

The multiple viewpoints endemic to product lines will surface in risk management and must be reconciled. For example, core asset developers may define success as meeting delivery dates, whereas product developers may define it as integrating the product and delivering capability to the end user. Both viewpoints are valid; the risk management process needs to "hear" both of them.

Sometimes two stakeholders on the same side of the product fence will have different viewpoints. For example, one military organization decided to use a product line approach to provide situational awareness tools to both embedded weapons systems and command and control systems. One embedded weapons system developer was concerned about weapons system safety and performance in a battle environment. This system developer could not accept immature products or those that weren't rigorously tested by the core asset developers. In the course of managing the project, the developer identified the following risk:

> *The core asset development team does not test the situational awareness core assets adequately; we may receive immature products with major defects that can impact the operational effectiveness of our weapons system.*

A command and control development team wanted increased functionality and was willing to accept early deliveries of "beta" version core assets. The team promised its customers incremental deliveries of new capabilities to support an upcoming operational test and identified the following risk:

> *The development process used by the core asset development team takes too long to support our time-to-field requirements; we may miss critical delivery commitments.*

These two risks, identified by separate product development teams with different project objectives, point to conflicting mitigation approaches. Using a team risk approach, each of these individual risks would be elevated to the product line organization level, and mitigation strategies would be developed to address both concerns. Those tasks are accomplished using a structure wherein personnel from multiple projects work together to share information about risks that may affect the other project(s) or the product line itself [Gallagher 1999a]. The risk at the product line organization level might be

The development process used by the core asset development team is rigid, forcing a "one-size-fits-all" delivery and integration approach; the delivery of core assets to reusers may not support project-level objectives and ultimately fail to meet the user's needs.

The complexity of managing the delivery of core assets to product development teams requires a management structure with mature methods and tools that can arbitrate conflicting needs and develop "win-win" solutions for the entire organization. Organizational risk management practices must provide proactive management methods and tools to help solve the complexities associated with implementing a product line approach.

Application to Core Asset Development

Core asset development teams should actively and continuously identify and manage risks associated with developing core assets for the product line. (For more information, see the "Technical Risk Management" practice area.) When the risks also affect product development teams or other core asset development teams, they should be elevated, using a team risk approach, to the organization level. Core asset development teams need to understand the objectives of the organization's product line approach and work closely with product developers, the developers of other core assets, and product line managers.

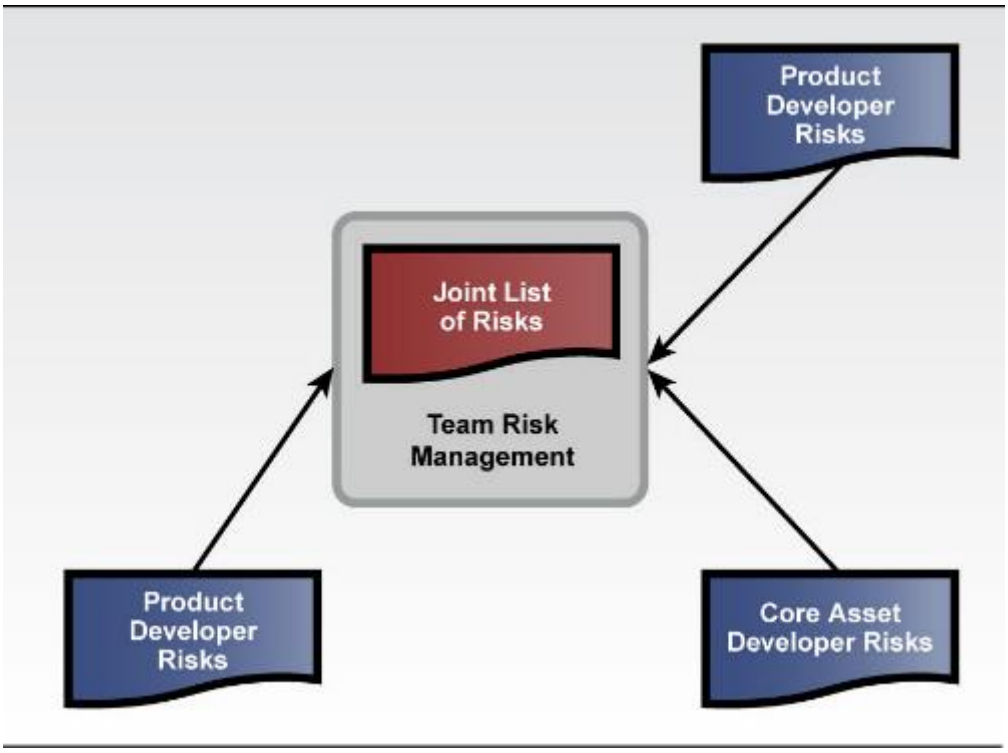
Application to Product Development

Product development teams should actively and continuously identify and manage risks on their projects. (For more information, see the "Technical Risk Management" practice area.) When risks cross project boundaries and affect other product development teams or core asset development teams, they should be elevated, using a team risk approach, to the organization level. Product development teams need to understand the objectives of the organization's product line approach and work closely with core asset development teams, other product developers, and product line managers.

Example Practices

The example practices outlined in the "Technical Risk Management" practice area apply here as well. The difference is that the scope of organizational risk management activities transcends individual development projects.

Team Risk Management (TRM): TRM, shown in the following figure, is one cross-organizational approach [Dorofee 1994a]. It was developed originally to allow individual and shared risk management between a government organization and a contractor organization. However, the basic principles are applicable to two projects or organizational structures that have mutual interests and shared risks. Simply stated, two or more projects could install risk management practices following the principles described in this practice area or the example practices described in the "Technical Risk Management" practice area. TRM describes how to establish an overarching risk management process to share risk management activities where appropriate and insure that project-level risks are surfaced at appropriately high levels of management.



Team Risk Management

Practice area risks: Risk management activities often gain a headstart by asking probing questions designed to test for the presence of risks that often occur in similar situations. In a product line organization, there are known risks associated with each practice area. Those risks can be used as the basis for probing questions associated with risk identification.

SEI Product Line Technical Probe (PLTP): The PLTP, described as an example practice under the "Launching and Institutionalizing" practice area, is a way surfacing a product line organization's strengths and challenges at an organizational level. When applied during a product line launch, the organizational risk management team should track and manage the uncovered challenges as risks.

Practice Risks

Non-conducive environment: The biggest risk in managing uncertainties is creating an environment in which team members feel that they can't talk about risks. In his book on software disasters, Glass points out that one shared characteristic of failed projects is the inability of project members to communicate potential problems to the decision makers within a project. Seventy-two percent of failed projects had team members who knew of impending doom, while only 19% of the project managers on the same projects shared their insights [Glass 1998a]. Risk management allows team members to discuss potential problems in a structured, non-threatening manner providing insight to decision makers.

The risks highlighted in the "Technical Risk Management" practice area also apply.

Structuring the Organization

This practice area is about how the organization forms groups to carry out the various responsibilities inherent in a software development effort. All organizations have a structure, even if it's only an implicit one, which defines the roles and responsibilities appropriate to the organization's primary activities. Particular organizational structures are chosen to accommodate enterprise goals and directives, the culture, the nature of business, and available talent. The organizational structure reflects the division of roles, responsibilities, and authority.

In a traditional organization (one that isn't oriented to product lines), individual product teams tend to be fully responsible for technical decisions that affect their products. Even in a matrixed organization, once engineers join a project from their specialty groups, the projects run independently. Specialization might require each product to have its own development group, with no sharing of personnel. The role of the organization's management in this case is to gather and allocate resources and provide high-level oversight–all with the end goal of supporting product teams.

Aspects Peculiar to Product Lines

A product line approach entails new roles and responsibilities related to the creation of core assets and of products from those assets. This practice area deals with placing those roles into the appropriate organizational units to most effectively support the product line approach.

In a product line context, the dual development of core assets and products dictates an organizational structure that is not product-centric. Beyond that, management must be concerned with identifying the organizational charter and boundaries, identifying functional groupings, allocating and assigning resources, monitoring organizational effectiveness, improving organizational operations, establishing interorganizational relationships, and managing organizational transitions.

Product line organizational structure goes hand in hand with product line operations (see the "Operations" practice area), because it is the embodiment of the roles, actors, and responsibilities defined there. Structuring the organization is like assembling a symphony orchestra–determining how many of each type of instrument will be in each section, selecting the musicians, and assigning people to the appropriate chair in their sections. "Operations" is the practice area that provides the music everyone will play together and guides them through performances. Both practice areas, obviously, are necessary. They result from an overall vision of how the product line will operate on a day-to-day basis. The job of organizational structure is to make sure that each responsibility and function of the product line operations has a work unit in which to reside.

An organizational structure should be chosen to assign at least the following tasks to the appropriate organizational unit or units:

- Determine the production strategy (see "Core Asset Development").
- Determine the product line scope and associated business case and refresh both in a routine and ongoing way.
- Produce and maintain the architecture for the product line.
- Determine the requirements for the product line and product line members.
- Design, produce, and maintain the product line's core assets and associated production plan.
- Assess the core assets for their utility and guide their evolution.
- Produce products.
- Determine the processes to be followed and measure or ensure compliance with them.
- Maintain the production environment in which products are produced.
- Forecast new trends, technologies, and other developments that might affect the future of the product line.

Application to Core Asset Development

A first-order choice that must be made when choosing an organizational structure is where to assign the people who develop, maintain, and evolve the product line's core assets. Typically, organizations take one of two approaches: (1) they form a separate unit for the developers and maintainers of core assets or (2) they house that effort in the same unit or units that build products [Bass 1997a, Bass 1998a].

Both approaches have their advocates. On the one hand, having a dedicated core asset development group makes the core asset developers one level removed from the pressure of project deadlines, making it more likely for them to produce assets that are generic and not too biased toward the individual product that happens to be under development at the time. Their loyalty is to the product line, not to any one product. On the other hand, having product engineers produce the core assets ensures that the core assets will be truly useful for products and ensures that people who build the core assets are not too far removed from the day-to-day realities of product production. In either case, there must be someone with a cross-product perspective and authority who can identify useful assets for the core asset base and encourage (if not direct) the appropriate product group to produce each core asset for use by other products.

When choosing whether to establish a separate core asset group, consider these factors:

- **the size of the effort and the number of products:** In a product line with many product groups and/or a large number of developers, distributing the core asset task results in an untenably high number of communication channels: every product group will have to talk to every other one. In this circumstance, it helps to have a dedicated core asset group.
- **new development or mostly legacy-based development:** In product line efforts where the core assets are built based on mining legacy systems, it makes more sense to have product developers (who will probably be more familiar with the legacy systems) be responsible for mining core assets that will be generic and fit the scope of the product line.
- **the funding model:** Funding a core asset development group can be problematic. Who pays for it? When working from legacy systems or when the product line

approach matures, it may be hard to justify a separate asset development group when the product development groups are adding product-specific features to the core assets.

- **the high or low effort of tailoring core assets:** How much development has to be done to get from the core assets to the products? If the amount of tailoring and new development is small, it may make sense to have most people work in a dedicated fashion on the core assets. If producing products requires substantial tailoring and new development, the asset development job is small by comparison, and integrated groups may be the answer.
- **the volatility of core assets:** Having core assets that evolve frequently and substantially argues for having a dedicated group to manage them, rather than overwhelming the product builders.
- p**arallel or sequential product development:** If products are built sequentially, it makes sense to have an integrated team working on them. When several product development projects are performed in parallel, there is a stronger need for a separate core asset group to avoid the multiple redevelopment of the same functionality.

An evolutionary organizational structure resolves the choice essentially by having it both ways. The approach starts the product line effort without a core asset group so that the engineers can assume the full responsibility for turning out a real product under the product line paradigm. However, as soon as there are parallel developments in progress, product development is separated from core asset development. Other approaches that seek a middle ground include frequent staff rotation and processes that require close communication among the groups.

Whichever way this issue is decided, the product line core assets must be managed to bring long-term benefit to the entire organization, rather than to just one specific application. The organizational structure chosen for product line production must assign the decision-making responsibilities for these functions.

And while managing the architecture and other reusable software-related core assets is an obvious product line responsibility that must be assigned to the right organizational unit(s), the management of non-software core assets must be allocated also to achieve the full benefit of core asset reuse. For example, where, how, and by whom will core assets such as product plans, schedules, and budgets be maintained for use across the product family?

Application to Product Development

Product line systems are developed and managed according to a life cycle that differs from the norm. Building products from core assets places greater emphasis on software integration and the testing of interfaces across components. For the unit or units assigned to product development, responsibilities include the following:

- making sure that each new product uses the core asset base according to the production plan
- working with the core asset owners to evolve new capabilities if the core assets are deficient in some way for a new product
- providing feedback to the core asset developers concerning the suitability and quality of the core assets

The product unit(s) may also negotiate customer requirements to situate new products within the scope of the product line to the greatest degree possible, although some organizational structures assign the management of product line requirements to their own units.

The organizational structure must assign responsibility for these roles, as well as for the more traditional product development roles.

Example Practices

Organizational models from a Swedish product line survey: Jan Bosch described four separate organizational models [Bosch 2000b] after studying a number of product line corporations. He discriminated between the models based only on organizational size and did not take into account other factors. The four models he identified were

- **development department:** In this model, all software development is concentrated in a single unit. Each member of the unit is expected to be a jack-of-all-trades in the product line, performing core asset development tasks or product development tasks when appropriate. This model appears in small organizations and those that provide consulting services. Although it is a simple model with short communication paths, it has a number of distinct drawbacks. Bosch wrote that it probably works only for units of up to 30 people. But in very small organizations whose product lines are commensurately small, it can be a viable starting-out approach.
- **business units:** Each business unit is responsible for a subset of the systems in the product line, which are clustered by similarity. Core assets are developed by the business units that need them and made available to the community; collaboration across business units to develop new core assets together is possible. This model has variations depending on how much flexibility a business unit has in developing (or modifying) a core asset. With no constraints, the products tend to spiral off on their own evolution paths, negating the product line approach. At higher levels of maturity, the responsibilities for particular core assets are assigned to specific business units. These units are constrained to maintain their core assets for the general use of the entire product line, and other business units are required to use them. Bosch estimates that this model could apply to organizations with 30 to 100 employees. However, this model suffers from the obvious risk, mentioned above, that a business unit will focus on its own product(s) first, and the good of the product line will take a back seat.
- **domain engineering unit:** In this model, a special unit is given the responsibility of developing and maintaining the core asset base. Business units build the products using those core assets. Bosch writes that when organizations exceed 100 employees, the *n*-to-*n* communication channels among separate business units become untenable, and a focusing channel to a central core asset unit becomes necessary. In this model, a strong and disciplined process becomes much more important in order to manage the communication and ensure that the overall health of the product line is the endgame of all parties.
- **hierarchical domain engineering units:** In a product line that is very large and/or very complex, it may pay to regard it hierarchically. That is, the product line may consist of subgroups that have more in common with each other than with other members of the product line. In this case, one core asset development unit may turn out core assets for the product line at large, and another may turn out core assets for the specialized subgroup. This example has only two levels, but the model could be extended indefinitely if the subgroup has a specialized subgroup within it and that specialized subgroup has another within it and so forth. This model works for very

large product lines built by very large organizations. Its main disadvantage is its tendency to bloat, reducing the organization's responsiveness to new needs.

Organizational structure for a reuse business: Jacobson, Griss, and Jonsson call for organizing a reuse-based business around a set of competence units, which contain "workers with similar competencies and entity object types that these workers are responsible for" [Jacobson 1997a, Ch. 9]. They prescribe the following units:

- requirements capture unit
- design unit
- testing unit
- component engineering unit
- architecture unit
- component support unit

The component support unit is responsible for "packaging and facilitating the reuse of component systems, and [is] mostly concerned with maintaining the facades and distributing the component systems so that reusers can access the desired components" [Jacobson 1997a].

Workers from these competence units are drawn to form an application family engineering team, one or more component system engineering teams, and a group of application system engineering teams. Application family engineering is where product line decisions are made such as which applications to develop and when. It also crafts the broad product line architecture. Component system engineering is responsible for the detailed architecture and design of a component system. Application system engineering builds products for individual customers.

Small, distributed core group with component representatives: In choosing whether the core asset group is separate or distributed throughout product groups, at least one company we know has opted for a hybrid approach. This company, which has about 50 employees building a product line of information systems, has a set of product groups that revolve around a small but central core group. In this model, the core group is responsible primarily for common support activities such as configuration management, maintenance of the core asset base, some process definition, and the like. An architect for the entire product line would also normally reside in this group. However, in this company, the architecture is not being evolved, and the architect has been reassigned. In each product group, a component representative has the joint responsibility of ensuring the use of core components and creating and evolving the software core assets. A strict protocol is enforced in which any candidate for inclusion in the core asset base must have at least two component representatives sponsoring it. This protocol ensures the asset's reusability across at least two products.

Structure for a distributed or globally dispersed organization: Work force distribution and globalization present additional challenges for organizing a product line effort. Some organizations are addressing these challenges in these ways:

- virtual core asset development teams with leaders in one location and dedicated core asset developers in other locations
- core asset development teams in one location and product development teams in other locations

Practice Risks

- **choosing a structure that is inappropriate for the organizational context at hand:** An inappropriate structure–one that doesn't match the talent base and/or culture of the organization–will cause the organization to fail to achieve the compromise between building overly generic core assets that are too general or expensive to serve any specific product and product-specific software that is too customized to serve in a product line.
- **lack of ample feedback and communication mechanisms:** Any organization's structure requires ongoing monitoring. These mechanisms are necessary to insure that the chosen structure is working as intended or to raise signal flags when it isn't.
- **one size fits all, for all time:** Many organizations report changing their organizational structure as their product line organizations mature. If your organizational structure is no longer adequate, it doesn't mean that it was wrong–it may just be a sign that it's time to change it.
- **reorganizing too often:** On the other hand, nothing seems to disrupt an organization like a reorganization. Make sure it is necessary before putting everyone through that stress.
- **ignoring key success factors:** Some key success factors in implementing structural change according to Myers, Maher, and Deimel[1] are described below. Failure to account for any of these factors constitutes a risk.
    - **the current level of organizational stress:** How much stress is the organization currently under as a result of previous changes or other stress-inducing factors?
    - **the implementation history:** How effective or ineffective has the organization been in effecting change in the past?
    - **sponsorship:** How effective is the key executive in obtaining commitments to change, communicating the support for change, and managing change?
    - resistance management: How will the organization address the inevitable resistance to change?
    - **culture:** Does the proposed change conflict or align with the organization's values, behaviors, and unwritten rules?
    - **change agent skills:** Are the change managers and staff equipped with the skills and motivation needed to implement the change?
- **adopting an organizational structure without an accompanying operational concept:** An organizational structure is just a wiring diagram. Without a clear articulation of the roles, responsibilities, and day-to-day operating procedures, any organizational structure will fail to meet its product line needs.

Technology Forecasting

Ironically, one of the most insidious dangers facing an organization is long-term success, for success breeds complacency. While routinely cranking out products and congratulating itself on its stellar productivity figures, an organization is vulnerable to being blindsided by a competitor who is sporting new features, new ideas, and new technologies. To head off such calamities, an organization must institutionalize vigilance, and one way to do that is by practicing technology forecasting. Technology forecasting helps provide strategic market planning–that is, identifying trends and predicting what the relevant markets will bear [Ryans 2000a]. It spots emerging standards, allowing an organization to position itself early to lead, or at least react, with agility. It reduces risk with respect to innovations, provides the basis for

planning and directing investments in research and development areas, and helps set the direction of product migration.

Technology forecasting helps take the pulse of the core technologies on which the products rely, as well as the tools, techniques, methods, and processes used to develop the products and bring them to market. Development techniques and tools are particularly important when time to market matters.

Technology is forecast in two areas:

1. technologies that support internal software development, which includes tools, processes, and methods for producing the software that will end up in products. These technologies may include
    - development paradigms and notations, such as the Unified Modeling Language (UML)
    - Web-based and wireless capabilities and languages such as the Extensible Markup Language (XML)
    - code development suites and environments
    - analysis tools
    - planning, configuration management, and requirements management techniques and tools
    - process improvement and process management approaches
2. customer solutions, meaning technologies that will affect (or end up as) features or capabilities embedded in products. These technologies may include
    - more efficient hardware platforms
    - better software platforms (such as databases or network and communications middleware)
    - improved user interfaces
    - faster database-searching strategies
    - new user-oriented paradigms (such as visual environments and Web-based interfaces)
    - improved architectural solutions
    - better problem-reporting systems
    - emerging standards
    - new features that enhance the capabilities of users

The rapid pace of technology change makes assessing new technology very challenging [Brown 1996a] and limits the technology time horizon to no more than three to five years. Consequently, plan to update your technology forecasts periodically. Christensen describes disruptive innovations–technologies that are not anticipated and have pervasive impact on entire markets [Christensen 1997a]. While disruptive innovations will likely elude technology-forecasting activities, most other technology changes can be anticipated by proactive looks across the domain and the software technology horizon.

Aspects Peculiar to Product Lines

For non-product-line development, technology forecasting may be performed infrequently. More typically, technology "now-casting" is done, meaning that the best currently available technologies are identified and assessed for their immediate applicability to the situation at hand and likely near-term changes. In product lines, however, technologies are identified and assessed continuously. They are assessed both for their immediate benefit and their potential

future applicability. A technology that shows promise today may find its way into the product line in three years, or it may flicker out before then. On the other hand, a technology deemed much too risky and on the edge today might be the foundation for the next revolution. In product line technology forecasting, time is an ally, and we can afford to be more patient than we could be in single-system development. And since we can amortize the cost over more products, we should also be able to be more thorough.

Technology forecasting for product lines covers both technologies that enable specific product features and technologies that support the engineering tasks in the development of those features. The former category will depend on the application domain and is too general for specific advice. The latter category includes

- software technology that supports variation modeling, automation, or producability
- process innovations that make the management, production, and deployment of new products or product releases more efficient and predictable
- configuration management strategies and techniques that help the product line organization provide better customer services through the ability to recreate and service the "instance" of the product installed and the upgrades that are appropriate for that installation
- trends within the relevant standards bodies and the migration of the technical basis of the standards that apply to the product line. Ideally, the product line organization exploits the technology forecast to achieve a technology leadership position.

The product line's technology forecasts are an important input into the development of the production strategy and the production method. These forecasts guide the development of forward looking strategies that define a migration path for integrating emerging new product construction technologies into the production method of the product line. The production strategy should be continually updated as technologies mature or fade and the long-term approaches to core asset and product development change. Revisions of the technology forecasts should signal the need to revisit the production strategy.

Application to Core Asset Development

Earlier in this practice area, we distinguished between technologies that benefit internal development and those that are customer focused. While both are relevant for core assets, the technologies that impact development require special emphasis as core assets. Tools and techniques, such as more efficient configuration management or improved tool integration, can be leveraged over a variety of products to improve efficiency and quality; these tools and techniques offer a strong strategic advantage. The architect and developers are primary stakeholders for these types of innovations.

Technology forecasts also directly aid the core asset designer in selecting variation points and variability mechanisms. The forecasts describe how radically technology is likely to change as well as which technologies will be most subject to change. The designer can determine how much flexibility is required at each variation point and select the variability mechanism accordingly. For example, minor changes that occur infrequently can be handled by a static mechanism such as class inheritance. Changes that occur more frequently may require a dynamic mechanism that can be automated or applied by the product user.

Application to Product Development

Technology forecasting increases the likelihood that products will be useful to customers by predicting those features that will become more (or less) desirable. The forecasts also help identify those features most likely to be affected by changes to their underlying implementation technologies. Both types of forecasts help identify variants that can be anticipated at variation points and ultimately when adjusting the product line scope.

Example Practices

Technology forecasting as continuous improvement: A continuous process improvement paradigm underlies many of the practices of technology forecasting. This paradigm suggests the need to constantly monitor the current technologies, tools, and processes that make up business practices in order to uncover opportunities for improved practices. The challenge is to focus the technology forecast on the areas offering the highest potential return to both the product line organization and the customer. Opportunities for improvement may result from the continuous analysis of defects and trouble reports yielding insights into the areas where the product line could leverage technology improvements. Additional opportunities for improvement may surface as a result of an analysis of changes in the marketplace that have implications on new technology.

Technical steering group: A technical steering group is a proven mechanism for keeping current with technology trends. This type of group comprises senior technical managers who analyze new trends, customer needs, and technologies. One organization we know also appointed a group of senior engineers as "technology stewards" and tasked them with accumulating and maintaining sufficient expertise in various technical areas so that they could become reliable forecasters.

Technology sources: Once the product line organization has a rationale and focus for conducting the technology forecast, the next task is to identify the sources of technology that are relevant to their needs. Sources include

- centers of technology investigation, such as the Software Engineering Institute (SEI), university laboratories, and corporate research and development establishments that publish their results
- research and development conferences, such as the Software Product Line Conference (SPLC), the International Conference on Software Engineering (ICSE), the conference on Object-Oriented Systems, Programming Languages, and Applications (OOSPLA), and others
- networks of professional associations and associates
- journals and periodicals

Validating the forecast: Once focused on the forecast objectives, specific steps include the following:

- Perform sufficient research to isolate the subset of promising technologies.
- Validate the technologies through simulations or models to focus on the most promising technology.
- Analyze and quantify the benefits for both developers and customers. Determine any adverse impacts.
- Conduct pilot tests as a proof of concept.

- Integrate the technology with other assets and then conduct testing.
- Provide adoption training and rollout.

Practice Risks

Inadequately forecasting technology results, predictably, in new and promising technologies passing you by. That, in turn, results in a product line that is obsolete before its time and opens the door for a more attentive competitor to steal your market. An inadequate technology forecast can result from

- **forecasting that is driven by technology rather than by business needs:** Analysts may become enamored by the technology and the search for it and lose sight of the corresponding business requirements.
- **ivory-tower forecasting:** Technology forecasting can become a "sandbox" or an "ivory tower" that is so removed from the day-to-day operations that crucial issues, objectives, and priorities get lost.
- **a lack of purpose:** If the problem that the product line organization is trying to solve is unknown, the search for new or emerging technology to maintain product line leadership has the potential to be limitless.

An inadequate forecast can result in

- **the wrong technology choice:** The wrong product gets built, and the market turns elsewhere for solutions. That can be caused by making investments in the wrong technology or an obsolete one or by simply choosing "what's good" over "what's popular."
- **an architecture that can't support new technology:** Radical changes in technology that the architecture can't accommodate will result in the obsolescence of the product line. The product line organization has a lot riding on its architectural decisions and needs as much advance warning as possible about any radical technological changes that could undermine them.

Training

Training is a core activity of any software development organization. The purpose of it is to provide the skills and knowledge needed to perform software management and technical roles. A training program involves identifying the training needed by the organization and the entities within it and then developing or procuring that training. Thus, as in many other practice areas, there is an initiation or planning phase followed by an execution phase. Training can be informal through mentoring or other on-the-job mechanisms or formal through classroom instruction or video sessions. Adequate resources and funding are needed if the training is to be effective and should be documented in a training plan.

Aspects Peculiar to Product Lines

Training is an element of both the initial product line adoption and the longer term product line evolution. This practice area focuses on the training practices that need to be instituted by management to ensure that the organizational units responsible for creating, fielding, and evolving the product line have properly trained personnel.

Management's support of training includes

- committing to an appropriate training plan
- ensuring that the plan is implemented and that the training is monitored for effectiveness
- ensuring that the product line training is consistent with and supportive of the overall product line adoption process or any process-improvement efforts

An organization's approach to training in product lines must focus on establishing a core competence in the creation and usage of core assets. Thus, it is not enough, for example, to send people to a course in object-oriented design or software reuse and then expect them to build product lines. All training must occur within the context of the organization's adoption plan for product line practices and address the skills needed by people for the new roles they will assume within the organization as it moves away from the single-system, project-centered view to the multi-system, product line view.

Product line training must be viewed as a strategic activity that should be planned accordingly. A training plan should align with the overall product line adoption plan and tie training goals to the business goals of the organization. For example, if a business goal is to reduce the time to market of products in the product line, any training in software reuse practices must emphasize the creation of specifically targeted reusable assets rather than an opportunistic reuse library. In the product line context, reuse is a means to an end, not an end in itself, and reuse training must focus on designing for commonality and controlling variability rather than on creating class libraries.

The appropriate product line training also depends on the current state and experience base of the organization. For example, an organization already fairly sophisticated about architecture and architecture-based design will have less of a need for training in those areas.

Application to Core Asset Development

Training for core asset development is primarily training in the software engineering practice areas. Any such training should be preceded by an introductory course that explains product line concepts in general and the organization's planned product lines in particular. The specific training associated with each software engineering practice (for example, training in a specific domain-analysis method or training in architecture definition) must be tied to the goal of creating a core asset base to support a product line. Similarly, training on the tools for representing and documenting the outputs of a domain analysis or a software architecture definition effort should focus on complementing the analysis and design skills of the core asset creators rather than their coding skills.

If externally available software (for example, COTS components, Web services, or open source software) is to be acquired as part of a core asset strategy, the training should focus primarily on how to choose and integrate software that makes sense for the product line. Similarly, if legacy software is to be repackaged as a core asset for the product line, the training should focus primarily on how to analyze its reusability rather than on how to "wrap" it for inclusion in a current product.

Finally, training materials and plans make first-class core assets themselves.

Application to Product Development

Product development training is the complement of core asset development training; its primary goal is to ensure that product developers know how to create products in the product line from the core asset base. It, too, should be preceded by an introductory course in software product lines with a particular focus on the organization's product line. The emphasis is on the effective use and reuse of core assets such as a domain model and architecture following the dictates of a production plan. The architecture for a single system, for example, is derived from (if not identical to) the product line architecture and not built from scratch; the training must emphasize the need to follow the production plan that was established for the product line. One of the major "themes" of the product development training should be that core asset usage is strategic in nature and that core assets that appear to be less than optimal for a *particular* product may be an optimal element of the *overall* product line strategy.

Another important aspect of product development training is getting people to follow the process defined for using core assets and correcting problems. For example, problems with core assets should always be referred to the core asset creators rather than the product developers. That way local "fixes" proposed for a particular customer can be assessed against the long-term needs of the product line (for example, the configuration management of core assets or control of variations).

Example Practices

Develop a training plan: The most important elements of product line training are the identification of training needs and the creation of a strategic training plan to meet those needs. This plan must identify the current skill gaps and determine the training requirements needed to fill them. It must also address how those skills will be established and maintained in the teams that build product lines: in-house courses, external courses, on-the-job training, mentoring, and so forth. Creating and implementing such a training plan is a key element of the cultural change needed for product line adoption.

Train people for the transition to a product line approach: Training to prepare people for the transition to product lines includes such elements as

- an introductory course on product line concepts and terminology
- an overview of the organization's current and planned product lines
- an overview of the proposed development process, including changes in existing processes, organizational structure, and roles
- a presentation of the concept of operations (CONOPS) for the product line to explain the goal state of the organization and the role of training in achieving that state
- training in specific product line practices or concepts. Here, software architecture deserves a special mention as a concept whose role and use should be emphasized, especially among the product builders. A familiarization course on the particular architecture being used as the foundation of the product line (including the ways it supports variability) is often most useful.
- training in the production techniques that automate the production of products from core assets, as described in the production plan
- training in supporting technologies (such as tools for representing and documenting the core assets)

Note that the first four items above are really about education rather than training and that even experienced practitioners may need to be reeducated about how their skills and roles apply in the product line context.

Examples of organizations creating educational and training materials specifically for software product lines include

- the Software Engineering Institute (SEI), which offers a software product line curriculum with courses covering introductory topics, product line adoption, product line development, and diagnosing an organization's readiness to adopt or ability to succeed with a product line approach (www.sei.cmu.edu/productlines/spl_curriculum.html)
- the Fraunhofer Institute for Experimental Software Engineering (IESE), which offers a set of services, based on the IESE Product Line Software Engineering (PuLSE) method, for setting up and running a product line organization (www.iese.fraunhofer.de/PuLSE/)

Training needs to be tailored to the specific processes and skills of the organization and to the product line adoption plan. Establishing a core competence in core asset creation and usage means understanding and applying new concepts, not getting high marks in a training class.[1] Any training in, for example, domain analysis, software architecture, object-oriented technology, model-driven development, design patterns and frameworks, specific programming languages, or specific development environments must be planned and implemented as part of the product line strategy and not as ends in themselves.

Implement the training plan: Implementing the training plan includes making decisions about the most effective way to deliver the training: classroom training, hands-on training, tutorials, workshops, pilot projects, mentoring, and so forth. An important decision in this regard is the scope of any planned in-house training and mentoring capability and the extent to which external courses and instructors will be used. In general, implementing the product line training plan involves some or all of the following choices:

- augmenting current training activities to support product lines
- replacing existing training activities
- adding new training activities

To ensure that the training meets the goals established for it in the training plan, it must be monitored and measured. Any lessons learned about the timeliness, relevance, level of difficulty, deficiencies, and effectiveness of the training should be collected from the trainers and trainees and incorporated into future training.

Practice Risks

An inadequate training program results in a staff that is ill-prepared to perform their jobs efficiently and effectively in the product line organization. Component developers, for example, who are not trained to create truly reusable assets will probably not do so and will consequently undermine the quality of the core assets.

Inadequate training can result from

- **a sink-or-swim approach:** In this approach, the organization assigns ill-prepared people to perform product line tasks and has to scramble later to fill the educational gap.
- **inappropriate focus:** Too much time spent training people on the tools, programming language, or development environment without the requisite product technology foundation will only succeed in helping to automate the wrong operation.
- **a lack of strategic investment:** Sacrificing vital training in order to meet current customer schedules and deliverables will, in the long run, result in staff who are ill-equipped to handle the needs of the product line operation.
- **a lack of the big picture:** If insufficient time is spent on communicating the product line vision and on building an awareness and acceptance of the product line, the rest of the product line training will fail to have a meaningful context.
- **inadequate training resources:** A lack of the necessary instructors, funding, classroom facilities, hardware, or software can derail any training plan.
- **a lack of coordination:** If the training plan is not coordinated with the overall product line adoption plan or process-improvement plan, the staff will probably become frustrated and overwhelmed.
- **a lack of training assessment:** If the effectiveness of the training is not monitored or measured, there will be no basis on which to predict the results or improve the training.

Frequently Asked Questions

1. **Concepts and Terminology**
    - Isn't a product line just the group of products produced by a single business unit or profit/loss center?
    - If a product line is a set of products, does that mean I have to build them all?
    - What's the difference between a domain and a product line? For instance, I know there is a telecommunications domain, and I've heard of a telecommunications product line. What's the difference?
    - Isn't software product line practice the same as single-system development with reuse?
    - Vendors and other developers issue subsequent releases of single products all the time, and have been doing so for years. Each release is built in the same way-you would say from the same core asset base. Technically speaking, what's the difference between a software product line and multiple releases of the same product?
    - Isn't product line practice just another name for domain engineering?
    - Isn't product line practice just another name for component-based development?

2. **Are Product Lines Right for My Organization?**
    - My organization is very small. Can we build a software product line?
    - My organization is very large. Can we build a software product line?
    - How can I make my organization build software product lines when people are busy in their own stovepipes? I don't have enough influence to change the way the organization does business.

- The SEI Capability Maturity Model Integration® models (CMMI®) puts important aspects of product line practice at Level 4. Do I have to be at Level 4 before I can hope to field a software product line?
- What are the various economic motivations for a software product line?
- How long before the approach pays off?
- We're doing okay. Why should I change the way I do business and undergo the upheaval that a product line will bring?
- Can you really be competitive in the marketplace with a software product line? Doesn't it take away your flexibility if you're locked into a product line and, say, an architecture?

3. **Exploring the Issues More Deeply**
   - Do successful software product line organizations have certain traits in common?
   - You have one practice area called 'Requirements Engineering' and another called 'Understanding Relevant Domains.' What's the difference between requirements analysis and domain analysis?
   - What is the relationship between process improvement and product line practice?
   - Must all products in the software product line share the same architecture? If my products have different architectures but make heavy use of other shared assets, isn't that a software product line?

4. **Using Software Product Lines with Other Approaches**
   - Can I use open source software packages in my product line?
   - Does the software product line approach work in a globalization environment?
   - Can a product line approach be compatible with agile development methods?
   - Is a model-driven development (MDD) approach compatible with the software product line strategy?
   - Does a system-of-systems (SoS) context rule out the use of software product lines?
   - Is there any connection between service-oriented architectures (SOAs) and software product lines?
   - How are SOA and software product line approaches alike?
   - How are SOA and software product line approaches different?
   - Can SOA and software product lines be used together?
   - Can the framework provide guidance for a move to SOA?

5. **Product Lines in the Context of Acquisition**
   - We're a U.S. government contractor, and our government customer wants to build a product line by buying core assets from us and then staging an open competition to build products using them. How can we possibly compete on the product-building side when all these new companies enter the fray and claim to be able to build products cheaper, because they didn't have to pay for the core assets?

- We work for a government contractor that wants to bid on a competition to build products based on core assets developed by another contractor. How can we possibly compete given the intimate knowledge possessed by the asset development contractor?
- I'm a government contractor, and the government wants me to supply reusable core components for other organizations to use in building a product line. Why should I open myself up to the legal liability? I'll be liable if they use my components incorrectly!
- What acquisition strategies should a DoD or government organization consider when acquiring a product line? And what is their potential impact on the acquiring organization?

6. **Getting Started**
    - All right, I want to start a software product line. What do I do first?
    - My company builds a broadly related group of products, each of which is a group of closely related products. Should I plan to build a single product line or a group of product lines?
    - I want to pilot software product lines in my organization. What are the criteria for a good pilot project?
    - Where is the resistance to adopting a product line approach usually found?
    - Where can I go to get more information?
    - Where can I read about other organizations that have successfully adopted the software product line approach?

Concepts and Terminology

Isn't a product line just the group of products produced by a single business unit or profit/loss center?

That is what some people mean when they use the term *product line* (see "A Note on Terminology"), but it's not what we mean. A product line is a set of products carefully chosen to take simultaneous advantage of commonality and market opportunities. These opportunities could span what is normally produced by a single business unit. For example, the software for commercial avionics and the software for military avionics could span separate business units in a company but be developed as a single product line by a software group that reports to both. Or, a single business unit may be responsible for developing and fielding more than one software product line. In general, a business unit is formed for organizational or financial reasons and may be responsible for (part of) one or more product lines.

If a product line is a set of products, does that mean I have to build them all?

No. While the term *product line* usually brings to mind a specific set of fielded products, you can think of a product line as defining a virtual set of products, each sharing a set of common, managed features. Only those products that satisfy some specific market need are actually built. The others represent untapped capability that you would be willing to build should the opportunity arise.

What's the difference between a domain and a product line? For instance, I know there is a telecommunications domain, and I've heard of a telecommunications product line. What's the difference?

A domain is a specialized body of knowledge, an area of expertise, or a collection of related functionality. A product line is the set of software-intensive systems sharing a common, managed set of features that satisfy particular market or mission needs. The telecommunications domain is a set of telecommunications problems, which in turn consists of other domains (switching systems, protocols, telephony, networks, etc.). A telecommunications product line is a specific set of systems that addresses some of those problems. Some people use the term *domain* to refer to a set of systems that employ knowledge in a particular domain, but really that's an incorrect use of the term. It's hard to imagine any nontrivial software system that doesn't encompass knowledge from several domains.

Isn't software product line practice the same as single-system development with reuse?

No. It differs in two fundamental ways. First, building a software product line is all about planning a plurality of products that will exist in the field and be maintained simultaneously, not just one that evolves over time. And second, the reuse that's involved is carefully planned, strategic, and applies across the entire set. In fact, one of the primary distinguishing characteristics of product line practice is preplanned reuse rather than ad hoc or one-time-only reuse. Software product line practice encourages choices and options that are optimized from the beginning for more than a single system.

Vendors and other developers issue subsequent releases of single products all the time, and have been doing so for years. Each release is built in the same way–you would say from the same core asset base. Technically speaking, what's the difference between a software product line and multiple releases of the same product?

There are some similarities, but the differences are acute enough to matter. The primary difference is that vendors who release subsequent versions try to retire earlier versions as soon as they can. Also, later versions are usually supersets of earlier versions. The latest version is always the one of interest. In a software product line, all versions are of interest, because they provide different functionality and quality attributes, each serving different market segments or missions.

Isn't product line practice just another name for domain engineering?

Domain engineering addresses only half of the problem–core asset development and acquisition. Product line practice addresses both domain engineering and product development using the core assets–or what has been called application engineering.

Isn't product line practice just another name for component-based development?

Software product lines certainly rely on a form of component-based development. The typical definition of component-based development involves the selection of components from a library or the marketplace to build products. Though the products in software product lines certainly are composed of components, these components are all specified by the product line architecture. Moreover, the components are assembled in a prescribed way; the prescription comes both from the architecture and the production plan. Software product lines involve the

strategic use of components. Component-based development usually is missing such a strategic and systematic approach.

Are Product Lines Right for My Organization?

My organization is very small. Can we build a software product line?

Yes. There is no reason why even a one-person software boutique cannot benefit from product line practices. Small organizations are well served by "lightweight" versions of the practices. And some of the organizational and role changes that come with product lines may, in fact, be easier to carry out in a smaller organization. Case studies show that small companies (even start-ups) have understood that in order to field a number of different systems, their severe resource constraints compelled them to exploit all possible commonality among the systems. They used a product line approach to leverage their small staffs and budgets across a successful blend of products. For an example of such a case study, see [Clements 2002c, Ch. 11].

My organization is very large. Can we build a software product line?

Yes, and there are many examples of success. The larger the organization, the more important it is to gain control over the costs and procedures for doing business and to effectively manage products.

How can I make my organization build software product lines when people are busy in their own stovepipes? I don't have enough influence to change the way the organization does business.

Nobody said it would be easy. It is necessary to make the business case for product lines (see the "Building a Business Case" practice area) and to find a champion above the level of all the stovepipe organizations. One of the missions of the SEI's Product Line Practice Initiative is to provide the ammunition needed to make the business case to potential champions. If case studies are documented and practices are defined within the context of a well-defined framework, convincing the decision makers should be easier. Also, many adoption strategies are possible. For example, you could find a part of the organization where people are open to cross-unit cooperation and begin building a small product line there. Incrementally grow the scope of the product line, and make sure you collect data to record the economies that you gain. The "Launching and Institutionalizing" practice area is especially relevant here.

The SEI Capability Maturity Model Integration® models (CMMI®) puts important aspects of product line practice at Level 4. Do I have to be at Level 4 before I can hope to field a software product line?

No, but certain Level-2 and -3 practices are key. For example, an organization must have well-developed configuration management and product planning skills before it can have much hope of fielding a product line. Moreover, successful product line practice requires process discipline. That's why "Process Discipline" is one of the practice areas. Organizations that are at Level 4 can transition to product line practices with fewer risks, but the risks can be managed at lower maturity levels as long as they are identified and mitigated.

What are the various economic motivations for a software product line?

Many organizations that have started software product lines in recent years have done so out of economic necessity. They have found that they simply could not continue to do business as they had in the past and still be competitive. Chief among the economic benefits are reduced time to market, greater market agility, opportunities for mass customization, and lower unit costs. These factors are driven by greater productivity from scarce or costly worker resources. Many organizations are finding that they simply cannot find enough qualified people to expand their business without embracing product line practices or afford to hire them even if they were available. In the realm of acquisition, the government may commission a product line to eliminate wasteful duplication among programs or to enable it to assemble more cost-effective systems by more easily obtaining products from a range of vendors. See the "Building a Business Case" practice area for more information.

How long before the approach pays off?

That's a hard question to answer, because it depends on too many organization-specific conditions. If you have decided to build products for which there is no market, no matter how efficiently you build them, you will not make money. However, a slightly different form of the question can be asked: How many products are required before building them as a product line is more cost-effective than building them as separate systems? The answer to that question lies somewhere in the neighborhood of two or three systems. That is, if your product set is expected to be populated by three or more systems, you're almost certainly better off to build them as a software product line than as separate systems. (See "It Takes Two," [Clements 2002c, p. 226].)

We're doing okay. Why should I change the way I do business and undergo the upheaval that a product line will bring?

First of all, it's not a foregone conclusion that product line practice brings upheaval. Some adoption strategies prescribe starting small and growing incrementally. New technologies are emerging that are allowing companies to quickly extract and manage commonality among systems they've already fielded; these systems can form the foundation for a product line that can be grown over time. But no organization should begin to build a product line without understanding the costs and benefits. Even if you're doing fine now, circumstances that motivate an organization are often based on long-term vision and not the status quo. You should also ask yourself whether your competition is standing still. The choice to adopt product line practices should be a strategy for achieving specific business goals. That's why it's important to build a business case before you decide to launch a product line.

Can you really be competitive in the marketplace with a software product line? Doesn't it take away your flexibility if you're locked into a product line and, say, an architecture?

Competitiveness should be sharply increased, because you're gaining the ability to respond quickly to opportunities that fall within the product line's scope–a response time sometimes measured in days instead of years. Also gained is increased ability to respond to user needs. What is lost is unbounded variability. The key is to understand which variations are needed and which variation mechanisms can best support them. Early steps are understanding the relevant domains, building a business case for a set of products, and product line scoping. If domain analysis has captured the requirements of the application domain, the product line has been scoped commensurately, and the architecture and other core assets reflect those needs through preplanned variation mechanisms, new products can be brought to market far faster than before with minimal loss of flexibility as compared to one-of-a-kind systems. However, a

product line organization needs to beware of complacency and should always keep an eye on the horizon for new technologies, new user needs, and new opportunities that might compel a shift in the product line's scope to keep it vigorous into the future. Finally, you're never "locked" into a product line, as the question implies. If a business case so warrants, you can always build a product outside the product line. For example, if an opportunity to enter a new market comes up, that first product can be used to test the waters and could even become the basis for a new product line.

Exploring the Issues More Deeply

Do successful software product line organizations have certain traits in common?

They do indeed. The best ones are comfortable with process discipline, have a strong and tireless champion for the approach in a position of leadership while the effort is being launched, and have long and broad experience in the application areas covered by the product line. Less tangibly but just as observable is one other aspect, something we've nicknamed the "e pluribus unum effect." *E pluribus unum* is Latin, of course, for "out of many, one." All the most successful product line organizations we've observed regard their primary mission as building and maintaining the product line (singular) or (equivalently) a production *capability*. Organizations still climbing the product line hill tend to describe themselves as turning out products (plural). The difference in mindset is subtle but powerful. Companies with the singular outlook realize that they have built a product capability that transcends any one product or even any several products. To them, turning out products is the easy part.

You have one practice area called 'Requirements Engineering' and another called 'Understanding Relevant Domains.' What's the difference between requirements analysis and domain analysis?

Remember that a domain is a specialized body of knowledge or area of expertise. Hence, domain analysis explores and captures the knowledge areas key to a product line and therefore tends to be broader in scope than requirements analysis. Requirements analysis usually focuses on specific applications. For example, domain analysis for a product line might identify areas of general commonality and variation across a body of relevant functionality (perhaps by examining and comparing legacy systems that have that functionality). Requirements analysis would identify a specific set of commonalities and variations for the family of products to be built.

What is the relationship between process improvement and product line practice?

Process improvement is broader in scope. Product line practice focuses on how to use a common set of core assets to modify, assemble, instantiate, or generate multiple products referred to as a product line. The focus is on improvement in product management. Process improvement addresses software engineering in general. While there is overlap between process improvement and product line practice in key practice areas, product line practices provide guidance for one particular approach to software development. However, experience shows that an organization with poorly defined software processes, or lacking the discipline to follow the processes it has defined, will not fare well in the transition to product line practice. Hence, at the very least, product line practice and process improvement need to be engaged hand in hand. As organizations improve their processes, they often enjoy increased productivity. Organizations with greater process maturity can continue to make productivity increases by turning their focus to product line practices.

Must all products in the software product line share the same architecture? If my products have different architectures but make heavy use of other shared assets, isn't that a software product line?

Only if that reuse is carried out in a "prescribed way," as required by the definition of a software product line. In all the software product lines we have studied, that prescription is most effectively carried out by using a common architecture where individual products either share the same architecture or permitted variations of the same architecture–an architecture we call the *product line architecture*. Variations might, for example, involve replacing one component with a similar one, instantiating a multiple component a different number of times, or exhibiting some subset of the overall architecture.

Using Software Product Lines with Other Approaches

Can I use open source software packages in my product line?

You can if all of the following conditions exist:

- You can live with not having any control over the release schedule of the open source package.
- The provided variation mechanisms of the open source package are appropriate for your need to integrate the package into your product line. Most open source packages have variation mechanisms built in to support the disparate needs of a broad user community. If the adaptations you have to make to integrate the package into your product line can be handled by those provided mechanisms, you'll be in good shape when you have to integrate a new release of the package. If you have to make more substantial changes in the package, be prepared to redo all of those changes with every new release. Resist the temptation to clone and own! Otherwise, you'll lose the benefits of having a user community.
- The maturity of the package is appropriate for your needs. There are very robust packages available but also an enormous number of alpha or beta releases. Some of those products are also abandoned and never reach a mature state. When selecting a package, do some research about the popularity of that package and find out whether an active user community exists. If it does, you most likely will get a package at least as robust as what you would expect from your own in-house development group.

Also be aware that some open source licenses are crafted in such a way that would require you to make your products, when based on open source packages, also open source. Read the license agreements very carefully!

Does the software product line approach work in a globalization environment?

Globalization has two meanings. The first, also called *internationalization* or *localization*, refers to making a software product or software-intensive system work correctly around the world. You can easily see that the software product line approach applies straightforwardly in this case. Different locale requirements correspond directly to product line variation points, and, in fact, a scoping exercise and a product line perspective on requirements can greatly help in identifying those variations. Identifying variations can result in each member of the family being as lean as possible and perhaps letting developers concentrate more on global features as well as locale-specific features.

The second meaning of globalization involves separate development groups located around the world cooperating productively and correctly to build software. In this context, the question above asks whether product lines can be developed globally. They can. It is necessary to lay some groundwork before effective distributed development can occur, and product lines are not immune from this need. For example, it is very useful to first establish such things as a common development environment, a common configuration management system, and a minimum set of common processes (or at least crisp process interface points), so the exchange of information and artifacts can flow smoothly across site boundaries. Just as in global development for single systems, architecture plays a central role. It crisply defines the work assignments and responsibility boundaries, in the system but also among the development parties. In a product line situation, the responsibilities will include the design and development of variation mechanisms for core assets.

Can a product line approach be compatible with agile development methods?

The short answer is yes, as demonstrated by the successful use of eXtreme Programming (XP) in Salion's product line effort [Clements 2002d]. However, the larger point is that the applicability of agile methods is more strongly determined by whether a project's characteristics align with a method's "home ground." (See the example practices under the "Process Discipline" practice area.)

Boehm and Turner advocate a pragmatic, risk-driven approach to choosing appropriate aspects from both plan-driven and agile methods [Boehm 2004b]. For projects whose characteristics stray from agility's home ground, it may still be possible to partition off portions where agile methods can flourish.

One challenge to agile methods' applicability is the principle of simple design, de-emphasizing the importance of software architecture. Within XP, this concept is known as "You Aren't Going to Need It" (YAGNI). As Boehm says, YAGNI works fine when future requirements are largely unpredictable but can be highly inefficient where there is a reasonable understanding of future needs. Because a product line approach inherently means you set out to understand future needs, and indeed base your business strategy on that understanding, you *are* going to need a sufficiently defined product line architecture. However, once it's developed, the software architecture contributes very well to a team's tacit knowledge and can serve as a basis for other agile practices (e.g., for development activities within a partitioned area of the architecture).

Is a model-driven development (MDD) approach compatible with the software product line strategy?

Yes. The software product line strategy is compatible with a variety of technical approaches loosely grouped under the "model-driven" category including MDD, model-driven architecture, and model-driven engineering. The software engineering practice areas in the framework include the activities of a traditional development process but do not constrain how those activities are implemented. The model-driven techniques emphasize the disciplined use of a variety of models, whose currency is maintained throughout the product life cycle. A model-driven approach to software products lines would emphasize automatic product derivation [McGregor 2005b]. Models would be constructed that are capable of expressing the commonality and variations inherent in the product line's scope. A new product's requirements would be translated into product-specific configurations of the product independent models that are maintained as core assets. Tool-supported translation, laid out in

a production plan, would be used to derive products. In fact, model-driven development could be thought of as a production strategy, as described in "Core Asset Development" of the framework.

Does a system-of-systems (SoS) context rule out the use of software product lines?

No. On the contrary, software product lines can help reduce the complexity of an SoS context. An SoS comprises independent, self-contained systems that, when taken as a whole, satisfy a specified need. Many software-intensive contexts today are SoSs. The complexity involved can be daunting. However, in many cases, there is considerable commonality among some of the self-contained systems. Suppose, for example, that the SoS involves 200 separate systems that must all interoperate. But suppose further (as is often the case) that there are some clusters within those 200 that have considerable commonality and whose variations could be handled economically. By making those clusters into software product lines, you can tame the interoperability issue into a smaller, more manageable set of interfaces and thereby reduce the complexity of the SoS. Moreover, the economic advantages and predictability associated with software product lines will be highly beneficial to the SoS in question.

Is there any connection between service-oriented architectures (SOAs) and software product lines?

SOAs and software product line approaches to software development share a common goal. They both encourage an organization to reuse existing assets and capabilities rather than repeatedly redeveloping them for new systems to achieve desired benefits such as productivity gains, decreased development costs, improved time to market, higher reliability, and competitive advantage.

How are SOA and software product line approaches alike?

Both approaches promote reuse by developing applications/products based on a set of reusable components. Those components are developed with well-defined interfaces and processes that specify how the components are to be used, which enables applications/products to be produced in less time.

Adopting either approach requires implementing similar organizational policies and practices necessary to adopt a new technology or a new way of doing business. SOA and software product lines share many of the same organizational issues such as planning, funding, tool support, training, and the need to change the organizational mindset towards reuse. When starting to use either approach, it is imperative for both SOA and software product line efforts to have organizational commitment and a champion.

Both approaches focus on identifying the application building blocks or reusable components associated with the application(s). In SOA, services represent the reusable building blocks. Core assets are the basis for production of products in a software product line. Separate teams may be employed to develop the reusable components and the applications. Small or pilot studies help develop skills to evolve the organization towards an SOA or software product line capability.

In both cases, the initial building blocks may come from legacy systems. Identifying and retrieving product line assets or services from existing systems in order to obtain the benefits of reuse are equally difficult. Documentation and tool support aid this effort.

Application/product development for both approaches is orchestrated in a similar manner. Inputs include the requirements for a particular application/product, reusable components, and the details of how these components are to be used to build the application/product.

How are SOA and software product line approaches different?

While the goals and the use of reusable components in the SOA and software product line approaches are very similar, the process by which the two compose systems are very different.

A software product line is, fundamentally, a set of related products. Each product is formed by taking applicable components from the base of common assets, tailoring them as necessary through preplanned variation mechanisms such as parameterization or inheritance, adding any new components that may be necessary, and assembling the collection according to the rules of a common, product-line-wide architecture under the auspices of a production plan. New or updated core assets are rolled back into the core asset base for future systems.

In SOA, it is not necessary for the reusable component(s) to come from a centralized, organization-controlled service base. Multiple organizations may provide the services leading to the possibility that services may change or disappear without notification. While multiple organizations can have responsibility for the core asset base in the software product line paradigm, that is not the usual case.

SOA addresses the issue of variation through orchestration (coordinating the participating services), service versioning, or extensible XML data types (a process of evolving from one format to another without requiring central control of the format).

The product line architecture is a software architecture that satisfies the needs of the products within the product line's scope. It is a key core asset. SOA is not the software architecture of the system; it is a design philosophy and an approach to software development where

- Services provide reusable functionality with well-defined interfaces.
- An SOA infrastructure enables discovery, composition, and invocation of services.
- Applications are built using functionality from available services.

The product line architecture establishes the quality goals for a system–its performance, reliability, modifiability, and so forth. Since SOA implementations may span enterprise boundaries, the quality attributes are dependent on the Quality of Service (QoS) of each of the included services. Desired end-to-end qualities may be achieved in specifically engineered applications. However, if the services are used in a different context, they may not meet the expected QoS. Service developers need to understand the functional and QoS requirements of potential service users.

In a software product line, an established process for updates to the core asset base is followed as the product line evolves, as more resources become available, as fielded products are maintained, and as technological changes or market shifts affect the product line's scope. Unlike SOA, the core assets in a software product line approach include non-software assets as well as software components. Product line requirements, domain models, test cases, and so on provide significant strategic advantage. Also included in the core asset base is a production plan prescribing how the products are produced from the core assets. SOA employs an SOA governance to facilitate planned reuse of services. SOA governance means the creation,

deployment, enforcement, and verification of policies throughout the entire life cycle of SOA artifacts. SOA governance platforms may provide easy service discovery through a centralized service registry and management tools for planned reuse (on the service level), such as tracking subscribers to services, negotiating service level agreements (SLAs), communicating change requests and actual changes in service interfaces and data types.

Can SOA and software product lines be used together?

It is possible to build a stand-alone (i.e., non-product-line) application using SOA and to build a software product line without using SOA. In that sense, the two approaches are independent. However, it is also possible to combine the two approaches. In particular, it is possible and feasible for an organization to use services as a reusable core asset with which to build products in a software product line. That is a focus of the "Using Externally Available Software" practice area. Also, service providers and SOA application developers could take a product line approach to the development of services.

Can the framework provide guidance for a move to SOA?

Organizations moving to an SOA approach can benefit from software product line information captured in other practice areas as well. The practice areas related to organizational management could help organizations understand important issues in adopting a new reuse-based technology. And those related to technical management and software engineering could help organizations analyze legacy systems, determine make/buy/mine/commission decisions, use existing available software, and understand risk. Documents similar to the product line adoption plan (describing the desired state of the organization and a strategy for achieving that state) and the product line production plan (which prescribes how the products are produced from the reusable components) provide excellent templates for organizations moving to planned, reuse-oriented environments.

Planned and systematic reuse, exploiting economies of scope, and other important software product line issues could feed into the SOA design philosophy to help manage the growth of services and application developments, understand the dependencies between services, and determine the impact of service changes to the applications. The software product line approach would help control complexities created by the combinatorics of services and applications that occur over the entire life cycle of a system.

Product Lines in the Context of Acquisition

We're a U.S. government contractor, and our government customer wants to build a product line by buying core assets from us and then staging an open competition to build products using them. How can we possibly compete on the product-building side when all these new companies enter the fray and claim to be able to build products cheaper, because they didn't have to pay for the core assets?

First, other contractors can't necessarily build derivative products any cheaper. At best, they can probably build them as cheaply, but even that is questionable. In practice, the core asset contractor actually may have an inside track on follow-on product development by virtue of its domain knowledge, expertise, and intimate knowledge of the core assets. Other contractors may have a significant, and potentially steep, learning curve with regard to understanding the functionality and technical characteristics of the core assets, how to appropriately augment them with product-specific assets, and how to integrate and test them to create a finished

product. In fact, it is usually those other contractors that are concerned about the unfair advantage the asset contractor has, because, under acquisition reform, contractor experience often plays a large part in the technical evaluation criteria. In fact, in some cases, the core asset contractor is excluded from bidding on follow-on product developments to avoid claims of having an unfair advantage.

What all this boils down to is the need to create a business strategy that effectively balances the interests of the core asset contractor and the acquisition organization (and other prospective product development contractors). The core asset contractor obviously has a vested interest in protecting the competitive edge it has in the marketplace. The core assets it owns are an instantiation of its domain knowledge and software expertise and may be a major factor in maintaining its competitive advantage. Accordingly, there are several options the core asset contractor, in conjunction with the acquisition organization, may want to pursue. They include

- selling the core assets outright to the government (with negotiated government-usage rights) while maintaining all commercial rights to the core assets
- licensing government usage of the core assets on a per-product (or per-asset) licensing basis (e.g., the government pays a license fee for each product developed using the core assets)
- negotiating a follow-on contract with the asset contractor to manage, sustain, and evolve the core assets and provide technical support for product development through the auspices of the acquisition organization
- opening up product development to competition through a leader/follower type of contract with the core asset contractor taking the leader role
- allowing the core asset contractor to compete for product development (the same way as any other contractor) or, alternatively, prohibiting the core asset contractor from competing on product development
- avoiding relying on any one contractor by letting an umbrella contract (competitively) to multiple contractors that allows them to compete and/or collaborate on the follow-on development of individual products using the core asset base; this would allow a greater number of contractors to participate in product development and may open up the door to allow the core asset contractor to compete.
- pursuing one, or a combination, of the above options

While there is no one answer to the question that is posed, the above list (which is non-exhaustive) demonstrates that there are many viable contracting options that can be pursued depending on the goals and acquisition strategy of the government agency and the business strategy of the core asset contractor.

Finally, if the core assets make it so straightforward to produce products based on them, the core asset contractor is in a perfect position to rapidly bring entirely new products to the commercial marketplace, thus taking advantage of the new production capability created courtesy of the government.

We work for a government contractor that wants to bid on a competition to build products based on core assets developed by another contractor. How can we possibly compete given the intimate knowledge possessed by the asset development contractor?

Combined with the previous question, this question shows that the grass is always greener on the other side of the fence. The fact that both of these questions are frequently asked suggests

that product lines do not particularly tilt the playing field in either direction. In fact, this question is no different than if you had asked "How can I hope to compete, because I didn't build the commercial off-the-shelf (COTS) software or the government-furnished equipment (GFE) that I'm required to use under this contract?" Contractors compete successfully under those conditions all the time. So the premise behind the question–that a contractor can only compete successfully if it is tasked with building *all* the software–is just not valid. A pragmatic answer is that your company is no worse off than if the product line strategy had not been pursued.

I'm a government contractor, and the government wants me to supply reusable core components for other organizations to use in building a product line. Why should I open myself up to the legal liability? I'll be liable if they use my components incorrectly!

Liability issues can be tricky. They are negotiated between the acquisition organization and the development contractor(s) and often involve legal counsel. However, in the case cited, the government contractor would not be directly liable for how another (third-party) government contractor uses its product–especially if it uses the product incorrectly. A contractor who develops components for a government agency is liable to the government–not to another government contractor–to the extent stipulated in the expressed warranties that are part of the contract. There may also be implied warranties of fitness for use for the particular purpose for which the government will use the items. Contracting officers have to consult with counsel prior to asserting any claim for breach of an implied warranty.

The important thing to remember is that liability issues are not unique to product lines. The government often contracts for a piece of equipment from one manufacturer and then provides it as government-furnished equipment (GFE) to another contractor to integrate into its final product. In such cases, the liability for the GFE items rest with the government. However, the government may have recourse to go back to the original development contractor to have a defect corrected, depending on the particular contractual warranties that were negotiated and agreed to by both parties.

For commercial items (i.e., nondevelopmental items), liability considerations (expressed and implied) are described in Part 12 (Acquisition of Commercial Items) of the Federal Acquisition Regulations [FAR 2005a]. General and specific liability considerations that apply to both commercial items and developmental items are described in Part 46 (Quality Assurance) of the FAR.

Specific solicitation provisions and contract clauses on warranties and liabilities that may apply are described in Part 52 of the FAR and Part 252 of the of the Defense Federal Acquisition Regulation Supplement [DFARS 1998a] and include the following sections:

- FAR: 52.246-18; Warranty of Supplies of a Complex Nature
- FAR: 52.266-19; Warranty of Systems and Equipment Under Performance Specifications or Design Criteria
- FAR: 52.246-23; Limitation of Liability
- FAR: 52.246-24; Limitation of Liability-High Value Items
- DFARS: 252.246.7001; Data Warranty

The bottom line is that, during the contract solicitation period, any contractor considering bidding on a government contract can formally request that the contracting officer define (in writing) the extent and limitation of the contractor's liabilities.

What acquisition strategies should a DoD or government organization consider when acquiring a product line? And what is their potential impact on the acquiring organization?

There are several alternative acquisition approaches that a program manager should consider when contemplating adopting a product line approach. Three such approaches are

- Commission a government organization to develop the product line. This strategy involves acquiring a completely government-owned product line using the in-house capabilities of a designated government acquisition organization.
- Commission a supplier to develop the product line. This strategy involves acquiring a complete product line production capability and developing derivative products through contracting with one or more suppliers.
- Commission a supplier to develop products using its own product line. This strategy involves acquiring products directly from a supplier who has an existing product line and a demonstrated capability to build derivative products.

The potential impact of these particular approaches on the acquisition organization is summarized in the table below.

Product Line Acquisition Approach	**Relative degree of organizational sophistication needed by acquirer**	**Relative degree of acquisition complexity**
1a. Development by acquisition organization	HIGH	LOW
1.b Development by acquisition organization and later transitioned to contractor	HIGH	MEDIUM
2.a Development involves one supplier	HIGH	HIGH
2.b Development involves multiple suppliers	HIGH+++	HIGH+++
3.a Single product acquired from supplier-owned product line	LOW	LOW
3.b Multiple products acquired from supplier-owned product line	LOW	MEDIUM

Bergey and Cohen describe these strategies in more detail and cite example product line applications [Bergey 2006a].

Getting Started

All right, I want to start a software product line. What do I do first?

Begin by learning. There are many resources available to you, many of which are available on or through the SEI's Web site. You can

- Read case studies and experience reports of organizations pursuing software product lines. They will help you lay out some specific goals for adopting a product line approach and gain an understanding of what might be involved in your particular

situation. Assign the most applicable one(s) as reading material for interested people in your organization.
- Get to know the SEI *Framework for Software Product Line Practice,*[SM] especially essential activities.
- Determine an approach for adopting software product line practices in your organization. You'll need to choose an adoption strategy. The "Launching and Institutionalizing" practice area lists several approaches. You can also use product line practice patterns, described in Chapter 7 of Software Product Lines: Practices and Patterns [Clements 2002c], in particular, the Adoption Factory Pattern [Northrop 2004a], to help with adoption and getting the product line started.
- Take one or more product line courses, such as those in the SEI Software Product Line curriculum.
- Start to build a business case for making the switch to software product lines.
- Get involved with the software product line community by participating in one of the conferences and workshops.

Once you've done some of this suggested homework, you can take the following steps in your organization:

- Conduct a product line diagnosis to help gauge your organization's strengths and weaknesses with respect to software product line capability. We recommend that you arrange for an SEI Product Line Quick Look or an SEI Product Line Technical Probe. Because the probe relies on interviews with many different people throughout the organization, it's also a good activity for getting people to think about the product line approach and the organization's goals for adopting it.
- Build a product line adoption plan for your organization. The Adoption Factory Pattern can be very helpful [Northrop 2004a]. The SEI Adopting Software Product Lines course provides useful guidance and support materials.
- Use the What to Build pattern to help lay out the scope and business case for your software product line [Clements 2002c].

Other steps are involved, but the ones listed above will get you off to a good start.

My company builds a broadly related group of products, each of which is a group of closely related products. Should I plan to build a single product line or a group of product lines?

This question can be answered by making a business case for each alternative and weighing the costs and benefits. The SEI Structure Intuitive Model for Product Line Economics (SIMPLE) can help with this [Clements 2005a]. We've seen this situation in practice many times. Often, the choice is to go with the single, large product line. Examples come from avionics, missile software, embedded engine controllers, and shipboard command and control systems. However, other organizations have chosen separate product lines or even hierarchical product lines. A hierarchical product line is essentially a product line of product lines. The decision depends on the amount of commonality that can be extracted from the broadly related group, how expensive it is to accommodate the needed variation, and how easy it is to communicate and cooperate across the different groups involved. If the products "live" in separate business units, for instance, the organization might find separate product lines to be more manageable. If you're just beginning to use the product line concept, it will be much easier to launch one than several, but having a scope that is too broad will jeopardize success with the product line approach.

I want to pilot software product lines in my organization. What are the criteria for a good pilot project?

The general criteria are the same as for any pilot project. It should be manageable but not too complex. It should be strategic but not so central that the failure of the pilot will bring down the organization. It should build on strengths rather than weaknesses. Specific to product lines, a pilot should be in an established and well-understood problem area and be led by some of the best innovators. Starting with a legacy system rather than starting from scratch makes sense if the legacy system is in good health (that is, if it's architecturally sound, well documented, and uses modern technologies). If an established core asset base is available to jump-start the product line, so much the better. Finally, the scope of the pilot should be narrow enough for results to be achieved and assessed quickly. See the "Launching and Institutionalizing" practice area for more information.

Should I plan to take the proactive approach or the reactive approach to my software product line?

As described in "All Three Together," the proactive approach involves building the core assets first and then using them to spin out products, whereas the reactive approach calls for having products first and then extracting the core assets from them. These approaches are two extremes of a spectrum of possibilities in between. A likely compromise is to develop some core assets fresh while producing others from an existing product or stable of products. Key determinants are how well you can predict the future and your available time and resources. If you can, with high confidence, map out the scope of your software product line in detail (describing the commonalities and variations that your products will require), proactively building the core assets to serve that scope will probably pay off in terms of a rapid product-production capability. If, on the other hand, your scope is defined only in more general terms, trying to build the core assets ahead of time will probably result in a large amount of rework and frustration. In that case, it is better to refine the core assets as you field more products and gain more knowledge about your market. The rule of thumb is to use whatever information you have when you have it.

Where is the resistance to adopting a product line approach usually found?

Although it can come from almost anywhere, it often shows up at the middle-management level. Many times, the folks in the trenches recognize the benefits of doing things the new way, because they're the ones tasked with implementing and re-implementing almost identical applications multiple times. Conversely, senior management tends to have the vision and see the financial bottom lines. That is by no means always the case, however. We also know of an organization in which senior management was preoccupied with buyouts and mergers, and middle management stepped in and championed the transition to product line practice. When senior management turned their attention inward, they discovered to their surprise that their company was building software in an entirely new way.

Where can I go to get more information?

The SEI's Web site on product line practice (http://www.sei.cmu.edu/productlines) is a good place to start. There are several books on software product lines, including *Software Product Lines: Practices and Patterns* [Clements 2002c], *Software Product-Line Engineering: A Family-Based Software Development Process* [Weiss 1999a], *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach* [Bosch 2000a], and *Software*

Product Lines in Action [van der Linden 2007a]. In addition, *Software Reuse: Architecture, Process, and Organization for Business Success* [Jacobson 1997a] is oriented towards projects employing large-scale strategic reuse and is compatible in many ways with product line practice.

Conferences on software product lines are also emerging, leaving papers about both theory and practice in their wake. The flagship gathering for the field is the International Software Product Line Conference.

Where can I read about other organizations that have successfully adopted the software product line approach?

Case studies are excellent resources to help you get started, because they show you how other organizations tackled the product line issues they faced. Case studies are listed on the SEI's Web site. In addition, you should visit the Software Product Line Hall of Fame, where software product lines of lasting community value are inducted at each Software Product Line Conference. Each inducted product line is described, and references for more information are given. Finally, some of the product line books mentioned above include several case studies of successful product lines, especially *Software Product Lines: Practices and Patterns* [Clements 2002c] and *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering* [van der Linden 2007a].

Glossary

acquisition	The process of obtaining products and services via a contract or license.
acquisition strategy	A plan of action for achieving a specific goal or result through contracting or licensing for products and services.
architectural view	A representation of a set of system elements and the relationships among them.
attached process	The process associated with a core asset that tells a product builder how the core asset will be used .in the development of products.
business case	A tool that helps one you make business decisions by predicting how they will affect an organization. Business cases are used among other things to determine if pursuing a product line approach will be beneficial and to determine if a given product line scope makes business sense.
commission	To contract with another party to build a product or provide a service.
component	A unit of software composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties [Szyperski 1998a].
concept of operations	Description of an organization's structure, roles, responsibilities, communication mechanisms, processes, practices, and policies that all

	detail the way the organization operates.
configuration management	A discipline for evaluating, coordinating, approving or disapproving, and implementing changes in the artifacts that are used to construct and maintain software systems. An artifact can be a piece of hardware or software or documentation.
core asset	A reusable artifact or resource that is used in the production of more than one product in a software product line. A core asset may be an architecture, a software component, a domain model, a requirements statement or specification, a document, a plan, a test case, a process description, or any other useful element of a software production process.
core asset base	The complete set of core assets associated with a given software product line.
customer interface	The description of an organization's connection to its customer(s) including the people involved, the information flow, the communication content, and any applicable polices and procedures.
development	A generic word used to describe how software comes to be.
domain	An area of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that area.
domain analysis	A process for capturing and representing information about applications in a domain, specifically common characteristics, variations, and reasons for variation.
domain understanding	Extensive insight and experience in the domains relevant to an organization's software and/or system endeavors.
externally available software	Existing software that can be used free, licensed, or purchased. The options for externally available software include commercial off-the-shelf (COTS) software, open source software, freeware, and Web-based services.
Framework for Software Product Line Practice	An online product line encyclopedia that describes the essential activities and practices in which an organization must be competent in order to reap the maximum benefit from fielding a software product line. The framework was developed and is maintained by the SEI.
market analysis	The systematic research and analysis of the external factors that determine the success of a product in the marketplace.
mining	Finding, analyzing, and rehabilitating a piece of an existing software system to serve in a new system for which it was not originally intended.
organizational management practice areas	Those practice areas necessary for orchestrating the entire software product line effort.

platform	A word some use to mean the software assets in a product line core asset base.
practice area	A body of work or a collection of activities that an organization must master to successfully carry out the essential work of a software product line.
product	Deployed software-intensive system or software.
product constraints	The set of common and variant features and behavioral attributes associated with the products in the product line scope.
product line	A set of products that share a common, managed set of features satisfying the needs of a particular market segment.
product line adoption	An organization's change to a software product line approach, which involves developing a core asset base, supportive processes, and organizational structures; developing products from that asset base in a way that achieves business goals; and preparing itself to institutionalize product line practices.
product line adoption plan	An organizational plan that describes how product line practices will be rolled out across the organization.
product line approach	The technical and business practices necessary to build a family of products as a software product line.
product line architecture	A core asset that is the software architecture for all the products in a software product line. A product line architecture explicitly provides variation mechanisms that support the diversity among the products in the software product line.
product line scope	A description of the products that will constitute the product line or that the product line is capable of including.
production plan	The guide to how products in the software product line will be constructed from the product line's core assets.
production constraints	Any restrictions on the timing, development environment, processes, or developer skills associated with development of the products in a software product line.
production capability	The core asset base, supportive processes, and tools that enable the development of the products in a software product line.
production method	The overall implementation approach that specifies the models, processes, and tools used in the attached processes across core assets.
production process	The process used for building all products in a software product line. The production process is defined by the set of attached processes with the necessary process "glue" to join them together into a coherent whole.
production strategy	The overall approach for realizing both the core assets and products in

	a software product line.
project	A temporary endeavor aimed at creating a unique product or service. Typically a project has its own funding, accounting, and delivery schedule.
requirements engineering	The use of systematic and repeatable techniques to elicit, analyze, specify, verify, and manage system requirements.
reuse	Using an item more than once.
scoping	An activity that bounds the behaviors and features of a system or set of systems. In a product line approach, scoping is the activity that defines the product line scope.
software architecture	Structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003a].
software engineering practice areas	Those practice areas necessary for applying the appropriate technology to create and evolve both core assets and products.
software product line	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 and that are developed from a common set of core assets in a prescribed way.
software product line practice pattern	A description of an organization's context, the product line problem it is trying to solve, and how a set of practice areas can be used in concert to solve the problem.
strategic reuse	Planned, systematic reuse that implements tightly connected business and technical strategies.
technical management practice areas	Those practice areas necessary for managing the creation and evolution of the core assets and the products.
technology forecasting	Looking at future technologies that will either support internal software development or affect features or capabilities embedded in an organization's products.

Bibliography

[Abowd 1996a]	Abowd, G.; Bass, L.; Clements, P.; Kazman, R.; Northrop, L.; & Zaremski, A. *Recommended Best Industrial Practice for Software Architecture Evaluation* (CMU/SEI-96-TR-025, ADA320786). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

[Albert 2002a]	Albert, C.; Brownsword, L.; Bentley, D.; Bono, T.; Morris, E.; & Pruitt, D. *Evolutionary Process for Integrating COTS-Based Systems (EPIC) Building, Fielding, and Supporting Commercial-off-the-Shelf (COTS) Based Solutions* (CMU/SEI-2002-TR-005, ADA408653). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.
[Alexander 1979a]	Alexander, C. *The Timeless Way of Building*. New York, NY: Oxford University Press, 1979.
[Alhir 2002a]	Alhir, Sinan Si. "Understanding the Unified Process (UP)." *Methods and Tools 10*, 1 (Spring 2002): 2-17.
[America 2000a]	America, P.; Obbink, H.; van Ommering, R.; & van der Linden, F. "CoPAM: A Component-Oriented Platform Architecting Method Family for Product Family Engineering," 167-180. *Software Product Lines: Proceedings of the First Software Product Line Conference* (SPLC1). Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic Publishers, 2000.
[Anastasopoulos 2000a]	Anastasopoulos, M. & Gacek, C. *Implementing Product Line Variabilities* (IESE-Report No. 089.00/E, V1.0). Kaiserslautern, Germany: Fraunhofer Institut Experimentelles Software Engineering, 2000.
[ANSI 1992a]	American National Standards Institute. *Guide for the Preparation of Operational Concept Documents* (ANSI/AIAA G-043-1992). Washington, DC: American National Standards Institute, 1992.
[AOSA 2007a]	Aspect-Oriented Software Association. Aspect-Oriented software Development Home Page (2007).
[Arango 1994a]	Arango, G. Ch. 2, "Domain Analysis Methods," 17-49. *Software Reusability*. Hemel Hempstead, England: Ellis Horwood, 1994.
[Ardis 2000a]	Ardis, M.; Dudak, P.; Dor, L.; Leu, W.; Nakatani, L.; Olsen, B.; & Pontrelli, P. "Domain Engineered Configuration Control," 479-494. *Software Product Lines: Proceedings of the First Software Product Line Conference* (SPLC1). Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic Publishers, 2000.
[Bachmann 2000a]	Bachmann, F.; Bass, L.; Chastek, G.; Donohoe, P.; & Peruzzi, F. *The Architecture Based Design Method* (CMU/SEI-2000-TR-001, ADA375851). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000.
[Bachmann 2005a]	Bachmann, F. & Clements, P. *Variability in Software Product Lines* (CMU/SEI-2005-TR-012, ADA450337). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
[Baldwin 2002a]	Baldwin, Carliss Y. & Clark, Kim B. "The Option Value of Modularity in Design-An Example from Design Rules, Volume 1: The Power of Modularity" (May 2002).
[Basili 1984a]	Basili, V. R. & Weiss, D. "A Methodology for Collecting Valid Software

	Engineering Data." *IEEE Transactions on Software Engineering SE-10*, 6 (November 1984): 728-738.
[Bass 1997a]	Bass, L.; Clements, P.; Cohen, S.; Northrop, L.; & Withey, J. *Product Line Practice Workshop Report* (CMU/SEI-97-TR-003, ADA327610). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997.
[Bass 1998a]	Bass, L.; Chastek, G.; Clements, P.; Northrop, L.; Smith, D.; & Withey, J. *Second Product Line Practice Workshop Report* (CMU/SEI-98-TR-015, ADA354691). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998.
[Bass 1999a]	Bass, L.; Campbell, G.; Clements, P.; Northrop, L.; & Smith, D. *Third Product Line Practice Workshop* (CMU/SEI-99-TR-003, ADA361391). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.
[Bass 1999b]	Bass, L. & Kazman, R. *Architecture-Based Development* (CMU/SEI-99-TR-007, ADA366100). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.
[Bass 2000a]	Bass, L.; Clements, P.; Donohoe, P.; McGregor, J.; & Northrop, L. *Fourth Product Line Practice Workshop Report* (CMU/SEI-2000-TR-002, ADA375843). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000.
[Bass 2003a]	Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice, Second Edition*. Reading, MA: Addison-Wesley, 2003.
[Batory 2005a]	Batory, Don. "Feature Models, Grammars, and Propositional Formulas," 7-20. *Proceedings of the 9th International Software Product Line Conference (SPLC 2005)* (Lecture Notes in Computer Science volume 3714). Rennes, France, September 26-29, 2005. New York, NY: Springer, 2005.
[Bayer 2000a]	Bayer, J.; Muthig, D.; & Widen, T. "Customizable Domain Analysis," 178-194. *Proceedings of the First International Symposium on Generative and Component-Based Software Engineering (GCSE '99)*. Erfurt, Germany, September 28-30, 1999. New York, NY: Springer, 2000.
[Bean 2000a]	Bean, J. "Use XML Even As It Changes." *Enterprise Development 2*, 2 (February 2000): 44-50.
[Beck 1994a]	Beck, K. & Johnson, R. "Patterns Generate Architectures," 139-149. *Proceedings of the Eighth European Conference on Object-Oriented Programming (ECOOP '94)*. Bologna, Italy, July 4-8, 1994. New York, NY: Springer-Verlag, 1994.
[Beck 1999a]	Beck, K. *Extreme Programming Explained*. Reading, MA: Addison-Wesley, 1999.
[Beck 2002a]	Beck, K. *Test-Driven Development: By Example*. Boston, MA: Addison-Wesley, 2002.

[Beizer 1990a]	Beizer, B. *Software Testing Techniques*. Boston, MA: International Thompson Computer Press, 1990.
[Berczuk 2003a]	Berczuk, Steve. *Software Configuration Management Patterns*. Boston, MA: Addison-Wesley, 2003.
[Bergey 1998a]	Bergey, J.; Clements, P.; Cohen, S.; Donohoe, P.; Jones, L.; Krut, B.; Northrop, L.; Tilley, S.; Smith, D.; & Withey, J. *DoD Product Line Practice Workshop Report* (CMU/SEI-98-TR-007, ADA346252). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998.
[Bergey 1999a]	Bergey, J.; Smith, D.; Weiderman, N.; & Woods, S. *Options Analysis for Reengineering (OAR): Issues and Conceptual Approach* (CMU/SEI-99-TN-014, ADA370600). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.
[Bergey 1999b]	Bergey, J.; Fisher, M.; & Jones, L. *The DoD Acquisition Environment and Software Product Lines* (CMU/SEI-99-TN-004, ADA244787). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.
[Bergey 2001a]	Bergey, J.; O'Brien, L.; & Smith, D. *Options Analysis for Reengineering (OAR): A Method for Mining Legacy Assets* (CMU/SEI-2001-TN-013, ADA395201). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
[Bergey 2002a]	Bergey, J.; O'Brien, L.; & Smith, D. "Using the Options Analysis for Reengineering (OAR) Method for Mining Components for a Product Line," 316-327. *Software Product Lines: Proceedings of the Second Software Product Line Conference (SPLC2)*. San Diego, CA, August 19-22, 2002. Berlin, Germany: Springer, 2002.
[Bergey 2003a]	Bergey, J.; O'Brien, L.; & Smith, D. *Application of Options Analysis for Reengineering in a Lead System Integrator Environment* (CMU/SEI-2003-TN-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
[Bergey 2004a]	Bergey, J.; Campbell, G.; Cohen, S.; Fisher, M.; Gallagher, B.; Jones, L.; Northrop, L.; & Soule, A. *Software Product Line Acquisition: A Companion to a Framework for Software Product Line Practice, Version 3.0*. (2004).
[Bergey 2006a]	Bergey, John & Cohen, Sholom. *Product Line Acquisition in a DoD Organization-Guidance for Decision Makers* (CMU/SEI-2006-TN-020, ADA447911). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006.
[Birk 2003a]	Birk, A.; Heller, G.; John, I.; Joos, S.; Muller, K.; Schmid, K.; & von der Massen, T. *Report of the GI Work Group "Requirements Engineering for Product Lines"* (IESE-Report No. 121.03/E, V1.0). Kaiserslautern, Germany: Fraunhofer Institut Experimentelles Software Engineering, 2003.
[Boeckle 2002a]	Boeckle, G.; Munoz, J.; Knauber, P.; Krueger, C.; Leite, J.; van der Linden, F.; Northrop, L.; Stark, M.; & Weiss, D. "Adopting and Institutionalizing a Product Line Culture,"49-59. *Software Product Lines: Proceedings of the*

	Second Software Product Line Conference (SPLC2). San Diego, CA, August 19-22, 2002. Berlin, Germany: Springer, 2002.
[Boehm 1981a]	Boehm, B. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.
[Boehm 1988a]	Boehm, B. W. "A Spiral Model of Software Development and Enhancement." *Computer 21*, 5 (May 1988): 61-72.
[Boehm 1989a]	Boehm, B. *IEEE Tutorial on Software Risk Management*. Piscataway, NJ: IEEE Computer Society Press, 1989.
[Boehm 2000a]	Boehm, B. "…And Very Few Lead Bullets, Either" [CD-ROM]. *Proceedings of Impacts 2000: The 15th Annual Software Engineering Symposium*. Washington, DC, September 18-21, 2000. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000.
[Boehm 2004a]	Boehm, Barry; Brown, A. Winsor; Madachy, Ray; & Yang, Ye. "A Software Product Line Life Cycle Cost Estimation Model," 156-164. *Proceedings of the International Symposium on Empirical Software Engineering (ISESE 2004)*. Redondo Beach, CA, August 19-20, 2004. Los Alamitos, CA: IEEE Computer Society, 2004.
[Boehm 2004b]	Boehm, B. & Turner, R. *Balancing Agility and Discipline: A Guide for the Perplexed*. Reading, MA: Addison-Wesley, 2004.
[Booch 1994a]	Booch, G. *Object-Oriented Analysis and Design with Applications*. Reading, MA: Addison-Wesley, 1994.
[Bosch 2000a]	Bosch, J. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Reading, MA: Addison-Wesley, 2000.
[Bosch 2000b]	Bosch, J. "Organizing for Software Product Lines," 117-134. *Proceedings of the 3rd International Workshop on Software Architectures for Product Families (IWSAPF-3)*. Las Palmas de Gran Canaria, Spain, March 15-17, 2000. Berlin, Germany: Springer, 2000.
[Bosch 2002a]	Bosch, J. "Maturity and Evolution in Software Product Lines: Approaches, Artifacts, and Organization," 257-271. *Software Product Lines: Proceedings of the Second Software Product Line Conference (SPLC2)*. San Diego, CA, August 19-22, 2002. Berlin, Germany: Springer, 2002.
[Brassard 2001a]	Brassard, M. & Ritter, D. *Sailing Through Six Sigma*. Marietta, GA: Brassard & Ritter, 2001.
[Brooks 1987a]	Brooks, F. "No Silver Bullet: Essence and Accidents of Software Engineering." *Computer 20*, 4 (April 1987): 10-19.
[Brown 1994a]	Brown, A. W.; Carney, D. J.; Morris, E. J.; Smith, D. B.; & Zarrella, P. F. *Principles of Case Tool Integration*. Oxford, U.K.: Oxford University Press, 1994.
[Brown 1994b]	Brown, A. W. "Why Evaluating CASE Environments is Different from Evaluating CASE Tools," 4-13. *Proceedings of the Third Symposium on*

	Assessment of Quality Software Development Tools. Washington, DC, June 7-9, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.
[Brown 1996a]	Brown, A. & Wallnau, K. "A Framework for Evaluating Software Technology." *IEEE Software 13*, 5 (September 1996): 39-49.
[Brown 1998a]	Brown, A. W. & Wallnau, K. C. "The Current State of CBSE." *IEEE Software 15*, 5 (September/October 1998): 37-46.
[Brownsword 1996a]	Brownsword, L. & Clements, P. *A Case Study in Successful Product Line Development* (CMU/SEI-96-TR-016, ADA315802). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.
[Bruckhaus 1996a]	Bruckhaus, T.; Madhavji, N. H.; Janssen, I.; & Henshaw, J. "The Impact of Tools on Software Productivity." *IEEE Software 13*, 5 (September 1996): 29-38.
[Budgen 2003a]	Budgen, David & Thomson, Mitchell. "CASE Tool Evaluation: Experiences from an Empirical Study." *The Journal of Systems and Software 67*, 2 (2003): 55-75.
[Burrows 2005a]	Burrows, Clive & Wesley, Ian. *Ovum Evaluates: Configuration Management*. London, UK: Ovum, Ltd., 2005.
[Buschmann 1996a]	Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; & Stal, M. *Pattern-Oriented Software Architecture: A System of Patterns*. New York, NY: John Wiley & Sons, 1996.
[Cagan 1992a]	Cagan, M. & Wright, A. *Requirements for a Modern Software Configuration Management System*. Irvine, CA: Continuous Software Corporation, currently Telelogic, Inc., 1992.
[CARDS 1994a]	Comprehensive Approach to Reusable Defense Software (CARDS). *Training Plan* (STARS-VC-B003/001/00). Reston, VA: Unisys Corporation, 1994.
[Carney 1997a]	Carney, D. *Assembling Large Systems from COTS Components: Opportunities, Cautions, and Complexities*. SEI Monographs on the Use of Commercial Software in Government Systems. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997.
[Carney 1998a]	Carney, D.; Brownsword, L.; & Oberndorf, T. "The Opportunities and Complexities of Applying Commercial Off-the-Shelf Components." *Crosstalk 11*, 4 (April 1998): 4-6.
[Carney 1998b]	Carney, D. "Evaluation of COTS Products: Some Thoughts on the Process" [online]. *SEI Interactive 1*, 2 (September 1998).
[Carney 1998c]	Carney, D. "COTS Evaluation in the Real World" [online]. *SEI Interactive 1*, 3 (December 1998).
[Charan 1999a]	Charan, R. & Colvin, G. "Why CEOs Fail." *Fortune 139*, 12 (June 21, 1999): 68-78.

[Charette 1989a]	Charette, R. *Software Engineering Risk Analysis and Management*. New York, NY: McGraw-Hill, 1989.
[Chastek 2001a]	Chastek, G.; Donohoe, P.; Kang, K.; & Thiel, S. *Product Line Analysis: A Practical Introduction* (CMU/SEI-2001-TR-001, ADA396137). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
[Chastek 2002a]	Chastek, G., ed. *Software Product Lines: Proceedings of the Second Software Product Line Conference (SPLC2)*. San Diego, CA, August 19-22, 2002. Berlin, Germany: Springer, 2002.
[Chastek 2002b]	Chastek, G. & McGregor, J. *Guidelines for Developing a Product Line Production Plan* (CMU/SEI-2002-TR-006, ADA407772). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.
[Chastek 2002c]	Chastek, Gary; Donohoe, Patrick; & McGregor, John D. *Product Line Production Planning for the Home Integration System Example* (CMU/SEI-2002-TN-029, ADA405846). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.
[Chastek 2003a]	Chastek, Gary & Donohoe, Patrick. *Product Line Analysis for Practitioners* (CMU/SEI-2003-TR-008). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
[Chastek 2004a]	Chastek, Gary; Donohoe, Patrick; & McGregor, John D. *A Study of Product Production in Software Product Lines* (CMU/SEI-2004-TN-012), Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.
[Christensen 1997a]	Christensen, Clayton M. *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*. Boston, MA: Harvard Business School Press, 1997.
[Clemen 1991a]	Clemen, R. T. *Making Hard Decisions: An Introduction to Decision Analysis*. Boston, MA: PWS-Kent Publishing Co., 1991.
[Clements 1998a]	Clements, P.; Bass, L.; Chastek, G.; Northrop, L.; Smith, D.; & Withey, J. *Second Product Line Practice Workshop Report* (CMU/SEI-98-TR-015, ADA354691). Colorado Springs, CO, November 1997. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998.
[Clements 1998b]	Clements, P. & Weiderman, N. *Report on the Second International Workshop on Development and Evolution of Software Architectures for Product Families* (CMU/SEI-98-SR-003, ADA346343). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998.
[Clements 2000a]	Clements, P. *Active Reviews for Intermediate Designs* (CMU/SEI-2000-TN-009, ADA383775). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000.
[Clements 2001a]	Clements, P.; Kazman, R.; & Klein, M. *Evaluating Software Architectures: Methods and Case Studies*. Boston, MA: Addison-Wesley, 2001.
[Clements 2002a]	Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. *Documenting Software Architectures: Views and Beyond*.

	Reading, MA: Addison-Wesley, 2002.
[Clements 2002b]	Clements, P. & Krueger, C. Two-part Point/Counterpoint column: "Being Proactive Pays Off" and "Eliminating the Adoption Barrier." *IEEE Software 19*, 4 (July/August 2002): 28-31.
[Clements 2002c]	Clements, P. & Northrop, L. *Software Product Lines: Practices and Patterns*. Boston, MA: Addison-Wesley, 2002.
[Clements 2005a]	Clements, Paul C.; McGregor, John D.; & Cohen, Sholom G. *The Structured Intuitive Model for Product Line Economics (SIMPLE)* (CMU/SEI-2005-TR-003). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
[Clements 2005b]	Clements, P.; Jones, L.; McGregor, J.; & Northrop, L. "Project Management in a Software Product Line Organization." *IEEE Software 22*, 5 (September/October 2005): 54-62.
[Clemons 1997a]	Clemons, Eric K. & Hitt, Lorin M. *Strategic Sourcing for Services: Assessing the Balance Between Outsourcing and Insourcing*. http://opim.wharton.upenn.edu/~clemons/files/outsourcing_v4_2.pdf (June 1997).
[Cohen 1991a]	Cohen, S. G.; Stanley Jr., J. L.; Peterson, A. S.; & Krut Jr., R. W. *Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain and Appendices A-I* (CMU/SEI-91-TR-028, ADA256590). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1991.
[Cohen 1996a]	Cohen, S.; Friedman, S.; Martin, L.; Royer, T.; Solderitsch, N.; & Webster, R. *Concept of Operations for the ESC Product Line Approach* (CMU/SEI-96-TR-018, ADA313952). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.
[Cohen 1998a]	Cohen, S. & Northrop, L. "Object-Oriented Technology and Domain Analysis," 86-93. *Proceedings of the Fifth International Conference on Software Reuse*. Victoria, B.C., June 2-5, 1998. Los Alamitos, CA: IEEE Computer Society Press, 1998.
[Cohen 1999a]	Cohen, S. *Guidelines for Developing a Product Line Concept of Operations* (CMU/SEI-99-TR-008, ADA367714). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.
[Cohen 2001a]	Cohen, S. *Case Study: Building and Communicating a Business Case for a DoD Product Line* (CMU/SEI-2001-TN-020, ADA395155). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
[Cohen 2003a]	Cohen, S. *Predicting When Product Line Investment Pays* (CMU/SEI-2003-TN-017, ADA418466). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
[Coplien 1998a]	Coplien, J.; Hoffman, D.; & Weiss, D. "Commonality and Variability in Software Engineering." *IEEE Software 15*, 6 (November/December 1998):

	37-45.
[Crosby 1979a]	Crosby, P. B. *Quality Is Free*. New York, NY: McGraw-Hill, 1979.
[Crossroads 2006a]	CM Crossroads. CM Crossroads Home Page. http://www.cmcrossroads.com/ (2007).
[Cruikshank 1998a]	Cruikshank, J. L. & Sicilia, D. B. *The Engine That Could: 75 Years of Value-Driven Change at Cummins Engine Company*. Boston, MA: Harvard Business School Press, 1998.
[Curtis 1992a]	Curtis, B.; Kellner, M.; & Over, J. "Process Modeling." *Communications of the ACM 35*, 9 (September 1992): 75-90.
[Cusumano 1991a]	Cusumano, M. A. *Japan's Software Factories*. New York, NY: Oxford University Press, 1991.
[Czarnecki 2005a]	Czarnecki. K. "Overview of Generative Software Development," 326-341. *Unconventional Programming Paradigms (UPP) 2004* (Lecture Notes in Computer Science volume 3566). Le Mont Saint Michel, France, September 15-17, 2004. Berlin, Germany: Springer-Verlag, 2005.
[Dabrowski 1993a]	Dabrowski, C. & Katz, S. *A Context Analysis of the Network Management Domain* (NISTIR 5309). Gaithersburg, MD: National Institute of Standards and Technology, 1993.
[Dabrowski 1994a]	Dabrowski, C. & Watkins, J. *A Domain Analysis of the Alarm Surveillance Domain V1.0* (NISTIR 5494). Gaithersburg, MD: National Institute of Standards and Technology, 1994.
[Dager 2000a]	Dager, J. "Cummins's Experience in Developing a Software Product Line Architecture for Real-Time Embedded Diesel Engine Controls," 23-45. *Software Product Lines: Proceedings of the First Software Product Line Conference (SPLC1)*. Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic Publishers, 2000.
[Dart 1991a]	Dart, S. "Concepts in Configuration Management Systems," 1-18. *Proceedings of the Third International Conference on Software Configuration Managemen*t. Trondheim, Norway, June 12-14, 1991. New York, NY: Association for Computing Machinery Press, 1991.
[Davis 1990a]	Davis, A. M. *Software Requirements: Analysis and Specification*. Englewood Cliffs, NJ: Prentice-Hall, 1990.
[DeBaud 1999a]	DeBaud, J. & Schmid, K. "A Systematic Approach to Derive the Scope of Software Product Lines," 34-43. *Proceedings of the 21st International Conference on Software Engineering (ICSE)*. Los Angeles, CA, May 16-22, 1999. Los Alamitos, CA: IEEE Computer Society, 1999.
[Del Rosso 2006a]	Del Rosso, C. "Continuous Evolution Through Software Architecture Evaluation: A Case Study." *Journal of Software Maintenance and Evolution Research and Practice 18*, 5 (September/October 2006): 351-383.

[Deming 1986a]	Deming, W. E. *Out of the Crisis*. Cambridge, MA: MIT Center for Advanced Engineering, 1986.
[Deschamps 1995a]	Deschamps, J. & Nayak, P. R. *Product Juggernauts*. Watertown, MA: Harvard Business School Press, 1995.
[DFARS 1998a]	U.S. Department of the Navy. *The Defense Federal Acquisition Regulation Supplement (DFARS)* (1998).
[Diaz 2005a]	Díaz, Oscar; Trujillo, Salvador; & Anfurrutia, Felipe I. "Supporting Production Strategies as Refinements of the Production Process," 210-221. *Proceedings of the 9th International Software Product Lines Conference (SPLC 2005)*. Rennes, France, September 26-29, 2005. New York, NY: Springer, 2005.
[DISA 1993a]	Defense Information Systems Agency Center for Information Management (DISA/CIM) Software Reuse Program. *Domain Analysis and Design Process, V1*. Arlington, VA: Defense Information Systems Agency Center for Information Management, 1993.
[DISA 1995a]	Defense Information Systems Agency. *US Army Space and Strategic Defense Command, Software Reuse Business Model*. Washington, DC: The Department of Defense, 1995.
[Donohoe 2000a]	Donohoe, P., ed. *Software Product Lines: Proceedings of the First Software Product Line Conference (SPLC1)*. Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic Publishers, 2000.
[Dorfman 1997a]	Dorfman, M. & Thayer, R. H. *Software Requirements Engineering*. Los Alamitos, CA: IEEE Computer Society Press, 1997.
[Dorofee 1994a]	Dorofee, A.; Walker, J.; Gluch, D.; Higuera, R.; Murphy, R.; Walker, J.; & Williams, R. *Team Risk Management Guidebook*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1994.
[Dorofee 1996a]	Dorofee, A.; Walker, J.; Alberts, C.; Higuera, R.; Murphy, R.; & Williams, R. *Continuous Risk Management Guidebook*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.
[Dumaine 1989a]	Dumaine, B. "How Managers Can Succeed by Speed." *Fortune 119*, 4 (February 1989): 54.
[Ecklund 1996a]	Ecklund, Jr., E.; Delcambre, L.; & Freiling, M. "Change Cases: Use Cases That Identify Future Requirements," 342-358. *Conference Proceedings of the OOPSLA 96*. San Jose, CA, October 6-10, 1996. San Jose, CA: ACM Press, 1996.
[Eisner 1994a]	Eisner, H. "Systems Engineering Sciences," 1312-1322. *Encyclopedia of Software Engineering, Volume 2*. New York, NY: John Wiley & Sons, 1994.
[Estublier 2005a]	Estublier, Jacky; Leblang, David; van der Hoek, Andre; Conradi, Reidar; Clemm, Geoffrey; Tichy, Walter; & Wiborg-Weber, Darcy. "Impact of Software Engineering Research on the Practice of Software Configuration

	Management." *ACM Transactions on Software Engineering and Methodology 14*, 4 (October 2005): 383-430.
[Etzioni 1964a]	Etzioni, A. *Modern Organizations*. Englewood Cliffs, NJ: Prentice-Hall, 1964.
[FAA 1995a]	Federal Aviation Administration (FAA) Office of Information Technology Integrated Product Team for Information Technology Services. Ch. 6, "The Business Case." *Business Process Improvement (Reengineering) Handbook of Standards and Guidelines*. Washington, DC: Federal Aviation Administration, 1995.
[Fairley 1994a]	Fairley, R. "Risk Management for Software Projects." *IEEE Software 2*, 3 (May 1994): 57-67.
[FAR 2005a]	U.S. Federal Regulation Acquisition Agency. *Federal Acquisition Regulation*. http://www.arnet.gov/far/index.html (2005).
[Faulk 1997a]	Faulk, S. R. "Software Requirements: A Tutorial," 128-149. *Software Requirements Engineering*. Los Alamitos, CA: IEEE Computer Society Press, 1997.
[Faulk 2000a]	Faulk, S.; Harmon, R.; & Raffo, D. "Value-Based Software Engineering (VBSE): A Value-Driven Approach to Product-Line Engineering," 205-224. *Software Product Lines: Proceedings of the First Software Product Line Conference (SPLC1)*. Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic Publishers, 2000.
[Favre 2003a]	Favre, Jean-Marie; Estublier, Jacky; & Sanlaville, Remy. "Tool Adoption Issues in a Very Large Software Company," 81-89. *ACSE 2003: Third International Workshop on Adoption-Centric Software Engineering* (CMU/SEI-2003-SR-004, ADA416604). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
[Finkelstein 1994a]	Finkelstein, A.; Kramer, J.; & Nuseibeh, B. *Software Process Modeling and Technology*. New York, NY: John Wiley & Sons, 1994.
[Finnegan 1997a]	Finnegan, P.; Holt, R.; Kalas, I.; Kerr, S.; Kontogiannis, K.; Muller, H.; Mylopolous, J.; Perelgut, S.; Stanley, M.; & Wong, K. "The Software Bookshelf." *IBM Systems Journal 36*, 4 (November 1997): 564-593.
[Fowler 1999a]	Fowler, P.; Middlecoat, B.; & Yo, S. *Lessons Learned Collaborating on a Process for SPI at Xerox* (CMU/SEI-99-TR-006, ADA373332). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.
[Fraunhofer 2006a]	Fraunhofer Institut Experimentelles Software Engineering (IESE). *Product Line Architectures*. http://www.iese.fraunhofer.de/fhg/Images/CC_PLA_Flyer_e_2006_03_tcm168-62107.pdf (2006).
[Fritsch 2004a]	Fritsch, C. & Hahn, R. "Product Line Potential Analysis," 228-237. *Proceedings of the Third Software Product Lines Conference (SPLC 2004)* (Lecture Notes in Computer Science volume 3154). Boston, MA, August

	30-September 2, 2004. New York, NY: Springer, 2004.
[FSF 2007a]	The Free Software Foundation. *The Free Software Foundation Home Page*. http://www.fsf.org/ (2007).
[Gaffney 1992a]	Gaffney, J. E. & Cruickshank, R. D. "A General Economics Model of Software Reuse," 327-337. *Proceedings of the 14th International Conference on Software Engineering (ICSE)*. Melbourne, Australia, May 11-15, 1992. New York, NY: ACM, 1992.
[Gallagher 1997a]	Gallagher, B. P.; Alberts, C. J.; & Barbour R. E. *Software Acquisition Risk Management Key Process Area (KPA)—A Guidebook V1.0* (CMU/SEI-97-HB-002, ADA328098). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997.
[Gallagher 1999a]	Gallagher, B. P. *Software Acquisition Risk Management Key Process Area (KPA)—A Guidebook Version 1.02* (CMU/SEI-99-HB-001, ADA370385). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.
[Gamma 1995a]	Gamma, E.; Helms, R.; Johnson, R.; & Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.
[Ganesan 2005a]	Ganesan, D. & Knodel, J. "Identifying Domain-Specific Reusable Components from Existing OO Systems to Support Product Line Migration," 27-36. *R2PL 2005-Proceedings of the First International Workshop on Reengineering Towards Product Lines* (CMU/SEI-2006-SR-002, ADA448167). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
[Ganesan 2006a]	Ganesan, D.; Muthig, D.; & Yoshimura, K. "Predicting Return-on-Investment for Product Line Generations," 13-22. *Proceedings of the 10th International Software Product Lines Conference* (SPLC 2006). Baltimore, MD, August 21-24, 2006. Los Alamitos, CA: IEEE Computer Society, 2006.
[Garlan 1995a]	Garlan, D.; Allen, R.; & Ockerbloom, J. "Architectural Mismatch: Why Reuse Is So Hard." *IEEE Software 12*, 6 (November 1995): 17-26.
[Gelenbe 1999a]	Gelenbe, E, ed. *System Performance Evaluation: Methodologies and Applications*. Boca Raton, FL: CRC Press, 1999.
[Geppert 2003a]	Geppert, B. & Weiss, D. "Goal-Oriented Assessment of Product-Line Domains," 180-188. *Proceedings of the Ninth International Software Metrics Symposium (METRICS'03)*. Sydney, Australia, September 3-5, 2003. Los Alamitos, CA: IEEE Computer Society, 2003.
[Glass 1998a]	Glass, R. L. *Software Runaways: Lessons Learned from Massive Software Project Failures*. Upper Saddle River, NJ: Prentice-Hall, 1998.
[Gleick 1987a]	Gleick, J. *Chaos: Making a New Science*. New York, NY: Penguin Books, 1987.

[Goldberg 1995a]	Goldberg, A. & Rubin, K. S. *Succeeding with Objects: Decision Frameworks for Project Management*. Reading, MA: Addison-Wesley, 1995.
[Grady 1992a]	Grady, R. B. *Practical Software Metrics for Project Management and Process Improvement*. Englewood Cliffs, NJ: Prentice-Hall, 1992.
[Grady 1997a]	Grady, R. B. *Successful Software Process Improvement*. Englewood Cliffs, NJ: Prentice-Hall, 1997.
[Graham 1998a]	Graham, I. *Requirements Engineering and Rapid Development: An Object-Oriented Approach*. Essex, England: Addison-Wesley, 1998.
[Griss 1994a]	Griss, M. *Software Reuse: Objects and Frameworks Are Not Enough* (HPL-95-03). Palo Alto, CA: Hewlett-Packard, 1994. http://www.hpl.hp.com/techreports/95/HPL-95-03.html
[Griss 1998a]	Griss, M. L.; Favaro, J.; & d'Alessandro, M. "Integrating Feature Modeling with the RSEB," 76-85. *Proceedings of the Fifth International Conference on Software Reuse*. Victoria, B.C., June 2-5, 1998. Los Alamitos, CA: IEEE Computer Society Press, 1998.
[Hammond 1999a]	Hammond, J. S.; Keeney, R. L.; & Raiffa, H. *Smart Choices: A Practical Guide to Making Better Decisions*. Boston, MA: Harvard Business School Press, 1999.
[Harel 1998a]	Harel, D. & Politi, M. *Modeling Reactive Systems with Statecharts: The STATEMATE Approach*. Reading, MA: Addison-Wesley, 1998.
[Hofmeister 2000a]	Hofmeister, N. S. *Applied Software Architecture*. Reading, MA: Addison-Wesley, 2000.
[Hollander 1999a]	Hollander, C. R. & Ohlinger, J. "CCT: A Component-Based Product Line Architecture for Satellite-Based Command and Control Systems," 201-206. *Proceedings of the Workshop on Object Technology for Product-Line Architectures*. Lisbon, Portugal, June 15, 1999. Bilbao, Spain: European Software Institute, 1999.
[Hotz 2006a]	Hotz, L.; Wolter K.; Krebs, T.; Deelstra, S.; Sinnema, M.; Nijhuis, J.; & MacGregor, J. *Configuration in Industrial Product Families - The ConIPF Methodology*. Amsterdam, The Netherlands: IOS Press, 2006.
[Huang 2003a]	Y. Huang, I. J. Taylor, D. W. Walker, and R. Davies, "Wrapping Legacy Codes for Grid-Based Applications," 139-145. *Proceedings of the 17th International Parallel and Distributed Processing Symposium (Workshop on Java for HPC)*. Nice, France, April 22-26, 2003. Los Alamitos, CA: IEEE Computer Society, 2003.
[Huff 1996a]	Huff, K. E. "Effect of Product Lines on Current Process Technology," 5-7. *Proceedings of the 10th International Software Process Workshop*. Dijon, France, June 17-19, 1996. Los Alamitos, CA: IEEE Computer Society, 1996.
[Humphrey	Humphrey, W. & Feiler, P. *Software Process Development and Enactment:*

1992a]	*Concepts and Definitions* (CMU/SEI-92-TR-004, ADA258465). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1992.
[Humphrey 1995a]	Humphrey, W. *A Discipline for Software Engineering*. Reading, MA: Addison-Wesley, 1995. (See page 4 for a definition of software process and pages 441-459 for information on process definition.)
[Humphrey 2000a]	Humphrey, W. "Justifying a Process Improvement Proposal." *news@SEI interactive 3*, 1 (March 2000).
[IEEE 1987a]	Institute of Electrical and Electronics Engineers. *IEEE Guide to Software Configuration Management* (IEEE Std 1042-1987). New York, NY: Institute of Electrical and Electronics Engineers, 1987.
[IEEE 1990a]	Institute of Electrical and Electronic Engineers. *IEEE Standard Glossary of Software Engineering Terminology* (IEEE Std 610.12-1990). New York, NY: Institute of Electrical and Electronics Engineers, 1990.
[IEEE 1996a]	Institute of Electrical and Electronics Engineers. *IEEE Recommended Practice for the Adoption of Computer-Aided Software Engineering (CASE) Tools* (IEEE Std 1348-1995). New York, NY: Institute of Electrical and Electronics Engineers, 1996.
[IEEE 1998a]	Institute of Electrical and Electronics Engineers. *Information Technology–Guideline for the Evaluation and Selection of CASE Tools* (IEEE Std 1462-1998). New York, NY: Institute of Electrical and Electronics Engineers, 1998.
[IEEE 1998b]	Institute of Electrical and Electronics Engineers. *IEEE Guide for Information Technology–Systems Definition–Concept of Operations (CONOPS) Document*(IEEE Std 1362-1998). New York, NY: Institute of Electrical and Electronics Engineers, 1998.
[IEEE 2000a]	Institute of Electrical and Electronics Engineers. *Recommended Practice for Architectural Description of Software-Intensive Systems* (IEEE Std 1471-2000). New York, NY: Institute of Electrical and Electronics Engineers, 2000.
[IEEE 2005a]	Institute of Electrical and Electronics Engineers. *IEEE Standard for Software Configuration Management Plans* (IEEE Std 828-2005). New York, NY: Institute of Electrical and Electronics Engineers, 2005.
[ISO 1995a]	International Organization for Standardization. *Information Technology—Guideline for the Evaluation and Selection of CASE Tools* [ISO/IEC 14102:1995(E)]. Geneva, Switzerland: International Organization for Standardization, 1995.
[ISO 1995b]	International Organization for Standardization & International Electrotechnical Commission. *Quality Management—Guidelines for Configuration Management* [ISO 10007:1995 (E)]. Geneva, Switzerland: International Organization for Standardization/ International Electrotechnical Commission, 1995.

[ISO 2007a]	International Organization for Standardization. *International Organization for Standardization Home Page*. http://www.iso.org/iso/en/ISOOnline.frontpage (2007).
[Jackson 2000a]	Jackson, M. *Problem Frames and Methods: Structuring and Analyzing Software Development Problems*. New York, NY: Addison-Wesley, 2000.
[Jacobson 1997a]	Jacobson, I.; Griss, M.; & Jonsson, P. *Software Reuse: Architecture, Process, and Organization for Business Success*. Reading, MA: Addison-Wesley Longman, 1997.
[Jandourek 1996a]	Jandourek, E. "A Model for Platform Development." *Hewlett-Packard Journal 47*, 4 (August 1996): 56-71.
[Jansen 2004a]	Jansen, Anton & Bosch, Jan. "Evaluation of Tool Support for Architectural Evolution," 375-378. *Proceedings of the 19th IEEE International Conference on Automated Software Engineering (ASE '04)*. Linz, Austria, September 20-24, 2004. Los Alamitos, CA: IEEE Computer Society Press, 2004.
[John 2006a]	John, I.; Knodel, J.; Lehner, T.; & Muthig, D. "A Practical Guide to Product Line Scoping." *Software Product Lines: Proceedings of the 10th International Software Product Line Conference (SPLC 2006)*. Baltimore, Maryland, August 21-24, 2006. Los Alamitos, CA: IEEE Computer Society, 2006.
[Jones 1996a]	Jones, L. & Northrop, L. "The Establishing Phase: Planning for Successful Improvement." *Software Process: Improvement and Practice 2*, 1 (March 1996): 51-53.
[Jones 1996b]	Jones, L.; Kasunic, M.; & Ginn, M. "Managing Technology Change: Implementing the SEI's IDEAL Model in a Less Than Ideal World," Track 5 [CD-ROM]. *Proceedings of the Eighth Software Technology Conference*. Salt Lake City, UT, April 21-26, 1996. Hill AFB, UT: Software Technology Support Center, in cooperation with Utah State University, Continuing Education, 1996.
[Jones 1999a]	Jones, L. & Northrop, L. "Software Process Improvement Planning," 1-24. *Proceedings of the European Software Engineering Process Group Conference*. Amsterdam, Netherlands, June 7-10, 1999. UK: European Software Process Improvement Foundation, 1999.
[Jones 2002a]	Jones, L. & Soule, A. *Software Process Improvement and Product Line Practice: CMMI and the Framework for Software Product Line Practice* (CMU/SEI-2002-TN-012, ADA403868). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.
[Jones 2004a]	Jones, Lawrence G. *Software Process Improvement and Product Line Practice: Building on Your Process Improvement Infrastructure* (CMU/SEI-2004-TN-044, ADA431119). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004.
[Jones 2005a]	Jones, L. & Northrop, L. *Software Product Line Adoption in a CMMI*

	Environment (CMU/SEI-2005-TN-028, ADA441309). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
[Kang 1990a]	Kang, K.; Cohen, S.; Hess, J.; Novak, W.; & Peterson, A. *Feature-Oriented Domain Analysis (FODA) Feasibility Study* (CMU/SEI-90-TR-021, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.
[Kang 1998a]	Kang, K.; Kim, S.; Lee, J.; Shin, E.; & Huh, M. "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures." *Annals of Software Engineering 5*, 5 (September 1998): 143-168.
[Kang 2002a]	Kang, K.; Lee, J.; & Donohoe, P. "Feature-Oriented Product Line Engineering." *IEEE Software 19*, 4 (July/August 2002): 58-65.
[Karlsson 1995a]	Karlsson, E., ed. *Software Reuse: A Holistic Approach*. Chichester, England: John Wiley & Sons, 1995.
[Kasse 2002a]	Kasse, T. *Action Focused Assessment for Software Process Improvement*. Norwood, MA: Artech House, 2002.
[Kazman 1998a]	Kazman, R.; Klein, M.; Barbacci, M.; Lipson, H.; Longstaff, T.; & Carrière, S. J. "The Architecture Tradeoff Analysis Method," 68-78. *Proceedings of the ICECCS*. Monterey, CA, August 10-14, 1998. Los Alamitos, CA: IEEE Computer Society Press, 1998.
[Kazman 1999a]	Kazman, R.; Barbacci, M.; Klein, M.; Carrière, S. J.; & Woods, S. G. "Experience with Performing Architecture Tradeoff Analysis," 54-63. *Proceedings of the 21st International Conference on Software Engineering (ICSE)*. Los Angeles, CA, May 16-20, 1999. New York, NY: ACM, 1999.
[Kazman 2000a]	Kazman, R.; Klein, M.; & Clements, P. *ATAM: Method for Architecture Evaluation* (CMU/SEI-2000-TR-004, ADA382629). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000.
[Kazman 2002a]	Kazman, R.; O'Brien, L.; & Verhoef, C. *Architecture Reconstruction Guidelines, Third Edition* (CMU/SEI-2002-TR-034, ADA412306). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
[Kenwood 2001a]	Kenwood, Carolyn A. *A Business Case Study of Open Source Software*. http://www.mitre.org/work/tech_papers/tech_papers_01/kenwood_software/index.html (2001).
[Keuffel 1999a]	Keuffel, W. "Planning for and Mitigating Risk." *Software Development 7*, 9 (September 1999): S1-S5.
[Kirkpatrick 1992a]	Kirkpatrick, R. J.; Walker, J. A.; & Firth, R. "Software Development Risk Management: An SEI Appraisal," 1-28. *Software Engineering Institute Technical Review '92* (CMU/SEI-92-REV). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1992.
[Klein 1999a]	Klein, M.; Kazman, R.; Bass, L.; Carriere, J., Barbacci, M.; & Lipson, H. "Attribute-Based Architecture Styles," 225-243. *Proceedings of the First*

	Working IFIP Conference on Software Architecture (WICSA1). San Antonio, TX, February 22-24, 1999. Boston, MA: Kluwer Academic Publishers, 1999.
[Knauber 2000a]	Knauber, P.; Muthig, D.; Schmid, K.; & Widen, T. "Applying Product Line Concepts in Small and Medium-Sized Companies." *IEEE Software 17*, 5 (September/October 2000): 88-95.
[Knodel 2005a]	Knodel, J. & Muthig, D. "Analyzing the Produce Line Adequacy of Existing Components", 37-45. *R2PL 2005-Proceedings of the First International Workshop on Reengineering Towards Product Lines* (CMU/SEI-2006-SR-002, ADA448167). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
[Kotonya 1996a]	Kotonya, G. & Sommerville, I. "Requirements Engineering with Viewpoints," 150-163. *Software Requirements Engineering, Second Edition*. Los Alamitos, CA: IEEE Computer Society Press, 1996.
[Krasner 1988a]	Krasner, G. E. & Pope, S. T. "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80." *Journal of Object-Oriented Programming 1*, 3 (August/September 1988): 26-49.
[Kruchten 1998a]	Kruchten, P. *The Rational Unified Process: An Introduction*. Reading, MA: Addison-Wesley, 1998.
[Kruchten 2004a]	Kruchten, Philipe. *The Rational Unified Process: An Introduction, Third Edition*. Boston, MA: Addison-Wesley, 2004.
[Krueger 2001a]	Krueger, C. "Easing the Transition to Software Mass Customization," 282-293. *Proceedings of the 4th International Workshop on Software Product Family Engineering*. Bilbao, Spain, October 3-5, 2001. New York, NY: Springer, 2001.
[Krueger 2002a]	Krueger, Charles W. "Variation Management for Software Production Lines," 37-48. *Software Product Lines: Proceedings of the Second Software Product Line Conference (SPLC2)*. San Diego, CA, August 19-22, 2002. Berlin, Germany: Springer, 2002.
[Krut 96]	Krut, R. & Zalman, N. *Domain Analysis Workshop Report for the Automated Prompt and Response System Domain* (CMU/SEI-96-SR-001, ADA311456). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.
[Kuusela 2000a]	Kuusela, J. & Savolainen, J. "Requirements Engineering for Product Families," 61-69. *Proceedings of the 22nd International Conference on Software Engineering (ICSE)*. Limerick, Ireland, June 4-11, 2000. New York, NY: ACM, 2000.
[Lee 2000a]	Lee, K.; Kang, K.; Koh, E.; Chae, W.; Kim, B.; & Choi, B. "Domain-Oriented Engineering of Elevator Control Software," 3-22. *Software Product Lines: Proceedings of the First Software Product Line Conference (SPLC1)*. Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic Publishers, 2000.

[Leon 2005a]	Leon, Alexis. *Software Configuration Management Handbook, Second Edition*. Norwood, MA: Artech House, Inc., 2005.
[Lim 1998a]	Lim, W. C. *Managing Software Reuse: A Comprehensive Guide to Strategically Reengineering the Organization for Reusable Components*. Upper Saddle River, NJ: Prentice-Hall PTR, 1998.
[Lewis 2005a]	Lewis, G.; Morris, E.; O'Brien, L.; Smith, D.; & Wrage, L. *SMART: The Service-Oriented Migration and Reuse Technique* (CMU/SEI-2005-TN-029, ADA441900). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
[Livesey 1997a]	Livesey, D. & Guinane, T. *Developing Object-Oriented Software: An Experience-Based Approach*. Upper Saddle River, NJ: Prentice-Hall, 1997.
[Loveland-Link 1999a]	Loveland-Link, J.; Barbour, R.; Krum, A.; & Neitzel, A. *Rollout and Installation of Risk Management at the IMINT Directorate, National Reconnaissance Office* (CMU/SEI-99-TR-009, ADA375848). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.
[Low 1999a]	Low, G. & Leenanuraksa, V. "Software Quality and CASE Tools," 142-150. *Proceedings of the Ninth International Workshop on Software Technology and Engineering Practice (STEP '99)*. Pittsburgh, PA, August 30-September 2, 1999. Los Alamitos, CA: IEEE Computer Society Press, 1999.
[Lundell 2004a]	Lundell, Björn & Lings, Brian. "Changing Perceptions of CASE Technology." *The Journal of Systems and Software 72*, 2 (2004): 271-280.
[Maccari 2000a]	Maccari, Alessandro & Riva, Claudio. "Empirical Evaluation of CASE Tools Usage at Nokia." *Empirical Software Engineering 5*, 3 (November 2000): 287-299.
[Maccari 2002a]	Maccari, Alessandro; Riva, Claudio; & Maccari, Francesco. "On CASE Tool Usage at Nokia," 59-68. *Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE '02)*. Edinburgh, UK, September 23-27, 2002. Washington, DC: IEEE Computer Society Press, 2002.
[Marttiin 1998a]	Marttiin, P. & Koskinen, M. Similarities And Differences of Method Engineering and Process Engineering Approaches. http://citeseer.ist.psu.edu/marttiin98similarities.html (1998).
[McAndrews 1993a]	McAndrews, D. R. *Establishing a Software Measurement Process* (CMU/SEI-93-TR-016, ADA267896). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.
[McDonald 1995a]	McDonald, M. & Dunbar, I. *Market Segmentation: A Step-by-Step Approach to Creating Profitable Market Segments*. Basingstoke, England: Macmillan Business, 1995.
[McFeeley 1996a]	McFeeley, R. *IDEAL: A User's Guide for Software Process Improvement* (CMU/SEI-96-HB-001, ADA305472). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

[McGregor 1999a]	McGregor, J. D. "Validating Domain Models." *Journal of Object-Oriented Programming 12*, 4 (July/August 1999): 12-17.
[McGregor 2001a]	McGregor, J. D. *Testing a Software Product Line* (CMU/SEI-2001-TR-022, ADA401736). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
[McGregor 2002a]	McGregor, J. D. "Building Reusable Test Assets for a Product Line," 345-6. *Proceedings of the 7th International ICSR Conference*. Austin, TX, April 15-19, 2002. Berlin, Germany: Springer, 2002.
[McGregor 2005a]	McGregor, J. D. Arcade Game Maker Pedagogical Product Line: Concept of Operations, Version 2.0. (2005).
[McGregor 2005b]	McGregor, J. *Preparing for Automated Derivation of Products in a Software Product Line* (CMU/SEI-2005-TR-017, ADA448223). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
[Meyer 1997a]	Meyer, M. H. & Lehnerd, A. P. *The Power of Product Platforms: Building Value and Cost Leadership*. New York, NY: The Free Press, 1997.
[Microsoft 2007a]	Microsoft Corporation. *Microsoft Home Page*. http://www.microsoft.com/en/us/default.aspx (2007).
[Mockus 1999a]	Mockus, A. & Siy, H. "Measuring Domain Engineering Effects on Software Change Cost," 304-311. *Proceedings of Metrics 99: Sixth International Symposium on Software Metrics*. Boca Raton, FL, November 4-6, 1999. New York, NY: IEEE Computer Society Press, 1999.
[Moore 1991a]	Moore, G. A. *Crossing the Chasm, Marketing and Selling Technology Products to Mainstream Customers*. New York, NY: Harper Business Publishing, 1991.
[Morris 1993a]	Morris, C. & Ferguson, C. "How Architecture Wins Technology Wars." *Harvard Business Review 71*, 2 (March-April 1993): 86-96.
[Moszkowski 1986a]	Moszkowski, B. *Executing Temporal Logic Programs*. New York, NY: Cambridge University Press, 1986.
[Muller 1988a]	Muller, H. & Klashinsky, K. "Rigi-A System for Programming-in-the Large," 80-86. *Proceedings of the 10th International Conference on Software Engineering (ICSE)*. Raffles City, Singapore, April 11-15, 1988. New York, NY: IEEE Computer Society Press, April 1988.
[Muller 2000a]	Muller, H.; Jahnke, J.; Smith, D.; Storey, M.; Tilley, S.; & Wong, K. "Reverse Engineering: A Roadmap," 47-60. *The Future of Software Engineering. Proceedings of the 22nd International Conference on Software Engineering*. Limerick, Ireland, June 4-11, 2000. New York, NY: ACM, 2000.
[Muller 2003a]	Muller, H.; Weber, A.; & Wong, K. "Leveraging Cognitive Support and Modern Platforms for Adoption-Centric Reverse Engineering (ACRE)," 30-35. *Third International Workshop on Adoption-Centric Software Engineering* (CMU/SEI-2003-SR-004). Pittsburgh, PA: Software

	Engineering Institute, Carnegie Mellon University, 2003.
[Musa 1999a]	Musa, J. *Software Reliability Engineering*. New York, NY: McGraw-Hill, 1999.
[Newell 1972a]	Newell, A. & Simon, H. A. *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall, 1972.
[Northrop 2004a]	Northrop, Linda. *Software Product Line Adoption Roadmap* (CMU/SEI-2004-TR-022, ADA431117). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004.
[O'Brien 2001a]	O'Brien, L. & Stoermer, C. "MAP-Mining Architectures for Product Line Evaluations," 35-44. *Proceedings of the Third Working IEEE/IFIP Conference on Software Architecture (WICSA1)*. Amsterdam, Netherlands, August 28-31, 2001. Los Alamitos, CA: IEEE Computer Society, 2001.
[O'Brien 2002a]	O'Brien, L.; Stoermer, C.; & Verhoef, C. *Software Architecture Reconstruction: Practice Needs and Current Approaches* (CMU/SEI-2002-TR-024, ADA407795). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.
[O'Brien 2003a]	O'Brien, L. & Stoermer, C. *Architecture Reconstruction Case Study* (CMU/SEI-2003-TN-008, ADA413856). Pittsburgh, PA: Engineering Institute, Carnegie Mellon University, 2003.
[O'Brien 2005a]	O'Brien, L.; Bass, L.; & Merson, P. *Quality Attributes and Service-Oriented Architectures* (CMU/SEI-2005-TN-014, ADA441830). Pittsburgh, PA: Engineering Institute, Carnegie Mellon University, 2005.
[Oberndorf 1998a]	Oberndorf, P. *COTS and Open Systems*. SEI Monographs on the Use of Commercial Software in Government Systems. Pittsburgh, PA: Software Engineering, Carnegie Mellon University, 1998.
[Ohlinger 2000a]	Ohlinger, J. CCT Lessons Learned: What We Did, Why We Did It, and What We Would Do Differently. http://sunset.usc.edu/GSAW/GSAW2000/pdf/Ohlinger.pdf (2000).
[OMG 1996a]	Object Management Group. *The Common Object Request Broker: Architecture and Specification, Revision 2.0* (97-02-25). Needham, MA: Object Management Group, Inc., 1996.
[OMG 2005a]	Object Management Group. *Software Process Engineering Metamodel Version 1.1*. http://www.omg.org/technology/documents/formal/spem.htm (January 2005).
[OMG 2007a]	Object Management Group. *UML Resource Page*. http://www.uml.org (2007).
[OSI 2007a]	The Open Source Initiative. *Open Source Home Page*. http://www.opensource.org/ (2007).
[Ossher	Ossher, Harold; Harrison, William; & Tarr, Perri. "Software Engineering

2000a]	Tools and Environments: A Roadmap," 261-277. *Proceedings of the Conference on the Future of Software Engineering*. Limerick, Ireland, June 4-11, 2000. New York, NY: ACM, 2000.
[Osterweil 1987a]	Osterweil, L. "Software Processes Are Software Too," 2-13. *Proceedings of the 9th International Conference on Software Engineering (ICSE)*. Monterey, CA, March 30-April 2, 1987. New York, NY: IEEE Computer Society Press, 1987.
[Park 1996a]	Park, R. E.; Goethert, W. B.; & Florac, W. *A. Goal-Driven Software Measurement—A Guidebook* (CMU/SEI-96-HB-002, ADA313946). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.
[Parnas 1972a]	Parnas, D. "Information Distribution Aspects of Design Methodology," 339-344. *Proceedings of the 1971 IFIP Congress*. Ljubljana, Yugoslavia, August 23-28, 1971. Amsterdam, Netherlands: North-Holland Publishing Company, 1972.
[Parnas 2001a]	Parnas, D. & Weiss, D. "Active Design Reviews: Principles and Practices," 132-136. Weiss, D. & Hoffman, D., eds. *Software Fundamentals: Collected Papers by David L. Parnas*. Reading, MA: Addison-Wesley, 2001.
[Paulk 1993a]	Paulk, M.; Weber, C. V.; Garcia, S. M; Chrissis, M.; & Bush, M. *Key Practices of the Capability Maturity Model, Version 1.1* (CMU/SEI-93-TR-025, ADA263432). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.
[Paulk 1995a]	Paulk, M. C.; Weber, C. V.; Curtis, B.; & Chrissis, M. B. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, MA: Addison-Wesley, 1995.
[Peterson 1991a]	Peterson, A. S. & Cohen, S. G. *A Context Analysis of the Movement Control Domain for the Army Tactical Command and Control System (ATCCS)* (CMU/SEI-91-SR-003, ADA248117). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1991.
[Porter 1980a]	Porter, M. *Competitive Strategy*. New York, NY: The Free Press, 1980.
[Poulin 1997a]	Poulin, J. S. *Measuring Software Reuse: Principles, Practices, and Economic Models*. Reading, MA: Addison-Wesley, 1997.
[Poulin 1997b]	Poulin, J. S. "The Economics of Software Product Lines." *International Journal of Applied Software Technology 3*, 1 (March 1997): 20-34.
[Powell 1996a]	Powell, A.; Vickers, A.; Williams, E.; & Cooke, B. Ch. 11, "A Practical Strategy for the Evaluation of Software Tools," 165-185. *Method Engineering: Principles of Method Construction and Tool Support–Proceedings of the IFIP TC8, WG8.1/8.2 Working Conference on Method Engineering*. Atlanta, GA, August 26-28, 1996. London, UK: Chapman & Hall, 1996.

[Prahalad 1990a]	Prahalad, C. & Hamel, G. "The Core Competency of the Corporation." *Harvard Business Review 68*, 3 (May-June 1990): 79-92.
[Pressman 1998a]	Pressman, R. *Software Engineering: A Practitioner's Approach, Fifth Edition*. New York, NY: McGraw-Hill Book Company, 1998.
[Prieto-Diaz 1991a]	Prieto-Diaz, R. & Arango, G. *Domain Analysis and Software Systems Modeling*. Los Alamitos, CA: IEEE Computer Society Press, 1991.
[Pronk 2000a]	Pronk, B. J. "An Interface-Based Platform Approach," 331-352. *Software Product Lines: Proceedings of the First Software Product Line Conference (SPLC1)*. Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic Publishers, 2000.
[Radice 1994a]	Radice, R. & Garcia, S. "Tutorial 4: Upgraded Integrated Process/TQM Tools: An Integrated Approach to Software Process Improvement," *The Sixth Annual Software Technology Conference Papers* [CD-ROM]. Salt Lake City, Utah, April 11-14, 1994. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1994.
[Ramesh 1997a]	Ramesh, B.; Stubbs, C.; Powers, T.; & Edwards, M. "Requirements Traceability: Theory and Practice." *Annals of Software Engineering 3*, 3 (September 1997): 397-415.
[Reifer 1997a]	Reifer, D. J. *Practical Software Reuse: Strategies for Introducing Reuse Concepts in Your Organization*. New York, NY: John Wiley & Sons, 1997.
[Robertson 1998a]	Robertson, D. & Ulrich, K. "Planning for Product Platforms." *Sloan Management Review 39*, 4 (Summer 1998): 19-31.
[Rolland 1997a]	Rolland, C. *A Primer For Method Engineering*. http://citeseer.ist.psu.edu/cache/papers/cs/29244/ ftp:zSzzSzsunsite.informatik.rwth-aachen.dezSzpubzSzCREWSzSzCREWS-97-06.pdf/ rolland97primer.pdf (1997).
[Ryans 2000a]	Ryans, A.; More, R.; Barclay, D.; & Deutscher, T. *Winning Market Leadership: Strategic Market Planning for Technology-Driven Businesses*. New York, NY: John Wiley & Sons, 2000.
[Schmid 2000a]	Schmid, Klaus. "Scoping Software Product Lines," 513-532. *Proceedings of the First Software Product Line Conference (SPLC1)*. Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic Publishers, 2000.
[Schmid 2001a]	Schmid, Klaus; Thiel, Steffen; Bosch, Jan; Johnsson, Susanne; Jaring, Michel; Thomé, Bernhard; & Trosch, Siegfried. *Scoping*. http://www.esi.es/esaps/public-pdf/CWD124-08-06-01.pdf (June 8, 2001).
[Schmid 2005a]	Schmid, K.; Krennrich, K.; & Eisenbarth, M. *Requirements Management for Product Lines: A Prototype* (IESE-Report 061.05/E). http://publica.fraunhofer.de/eprints/N-31550.pdf (2005).
[Schmid 2006a]	Schmid, K.; Krennrich, K.; & Eisenbarth, M. "Requirements Management for Product Lines: Extending Professional Tools," 113-122. *Proceedings of*

	the Tenth International Software Product Line Conference (SPLC 06). Baltimore, Maryland, August 21-34, 2006. Los Alamitos, CA: IEEE Computer Society, 2006.
[Schmidt 2000a]	Schmidt, Douglas; Stal, Michael; Rohnert, Hans; & Buschmann, Frank. *Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects*. New York, NY: Wiley, September 2000. http://www.wiley.com/WileyCDA/WileyTitle/productCd-0471606952.html
[Schmidt 2003a]	Schmidt, M. J. *What's a Business Case? And Other Frequently Asked Questions*. http://www.solutionmatrix.com/downloads/Whats_a_Business_Case.pdf (2003).
[Schnell 1996a]	Schnell, K.; Zalman, N.; & Bhatt, A. *Transitioning Domain Analysis: An Industry Experience* (CMU/SEI-96-TR-009, ADA310918). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.
[Schwaber 2005a]	Schwaber, Carey with the assistance of Barnett, Liz; Friedlander, David; & Hogan, Lindsey. *The Expanding Purview of Software Configuration Management*. Cambridge, MA: Forrester Research, Inc. http://www.forrester.com/Research/Document/Excerpt/0,7211,36337,00.html (2005).
[Seacord 2001a]	Seacord, R.; Comella-Dorda, S.; Lewis, G.; Place, P.; & Plakosh, D. *Legacy System Modernization Strategies* (CMU/SEI-2001-TR-025, ADA396051). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
[Seacord 2003a]	Seacord R., Plakosh D., Lewis G. A. *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices*. Boston, MA: Addison-Wesley, 2003 (ISBN 0-321-11884-7).
[SEI 1995a]	Software Engineering Institute. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, MA: Addison-Wesley, 1995.
[SEI 2000a]	Software Engineering Institute. *Capability Maturity Model Integration, Version 1.1 CMMI for Systems Engineering and Software Engineering (CMMI-SE/SW, V1.1)* (CMU/SEI-2000-TR-018, ADA388775). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000.
[SEI 2006a]	Software Engineering Institute. *CMMI® for Development, Version 1.2 CMMI-DEV, V1.2* (CMU/SEI-2006-TR-008, ADA455858). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006.
[SEI 2007a]	Software Engineering Institute. *Attribute-Driven Design Method* (2007).
[SEI 2007b]	Software Engineering Institute. *Software Architecture for Software-Intensive Systems* (2007).
[SEI 2007c]	Software Engineering Institute. *COTS-Based Systems (CBS) Initiative* (2007).

[SEI 2007d]	Software Engineering Institute. *Product Line Analysis* (2007).
[SEI 2007e]	Software Engineering Institute. *Product Line Technical Probe* (2007).
[SEI 2007f]	Software Engineering Institute. *Reengineering* (2007).
[SEI 2007g]	Software Engineering Institute. *Quality Attribute Workshops* (2007).
[SEI 2007h]	Software Engineering Institute. *Economics of Software Product Lines* (2007).
[SEI 2007i]	Software Engineering Institute. *Software and Systems Process Improvement Networks (SPINs)* (2007).
[Shaw 1996a]	Shaw, M. & Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. Englewood Cliffs, NJ: Prentice-Hall, 1996.
[Shaw 2000a]	Shaw, J. *Control Channel Toolkit: Open Architecture-Based Product Line Development* http://sunset.usc.edu/GSAW/GSAW2000/pdf/shaw.pdf (2000).
[Sheard 1997a]	Sheard, S. "The Frameworks Quagmire." *Crosstalk 10*, 9. http://www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1997/09/frameworks.asp (September 1997).
[Shehata 2002a]	Shehata, M.; Eberlein, A.; & Hoover, J. *Requirements Reuse and Feature Interaction Management*. http://www2.enel.ucalgary.ca/People/eberlein/publications/FI_ICSSEA2002.pdf (2002).
[Six Sigma 2007a]	iSixSigma. *Six Sigma Home Page*. http://www.isixsigma.com/ (2007).
[Smith 1990a]	Smith, C. *Performance Engineering of Software Systems*. Reading, MA: Addison-Wesley, 1990.
[Smith 2001a]	Smith, C. & Williams, L. *Software Performance Engineering for Object-Oriented Systems*. Reading, MA: Addison-Wesley, 2001.
[Smith 2002a]	Smith, D.; O'Brien, L.; & Bergey, J. "Using the Options Analysis for Reengineering (OAR) Method for Mining Components for a Product Line," 316-327. *Software Product Lines: Proceedings of the Second Software Product Line Conference (SPLC2)*. San Diego, CA, August 19-22, 2002. Berlin, Germany: Springer, 2002.
[Sneed 2001a]	Sneed, H. M. "Recycling Software Components Extracted From Legacy Programs," 43-51. *Proceedings of the 4th International Workshop on Principles of Software Evolution*. Vienna, Austria, September 10-11, 2001. New York, NY: ACM Press, 2001.
[Sommerville 1997a]	Sommerville, I. & Sawyer, P. *Requirements Engineering: A Good Practice Guide*. New York, NY: John Wiley & Sons, 1997.
[Soni 1995a]	Soni, D.; Nord, R. R.; & Hofmeister, C. "Software Architecture in Industrial

	Applications," 196-207. *Proceedings of the 17th International Conference on Software Engineering (ICSE)*. Seattle, Washington, April 23-30, 1995. New York, NY: Association for Computing Machinery Press, 1995.
[SPC 1993a]	Software Productivity Consortium. *Reuse Adoption Guidebook* (SPC-92051-CMC, Version 02.00.05). Herndon, VA: Software Productivity Consortium, 1993.
[SPC 1993b]	Software Productivity Consortium. *Reuse-Driven Software Processes Guidebook* (SPC-92019-CMC, Version 02.00.03). Herndon, VA: Software Productivity Consortium, 1993.
[SPLiT 2004a]	International Workshop on Software Product Line Testing. *SPLiT Proceedings and Presentations*. http://www.biglever.com/split2004/presentations.html (2004).
[SPLiT 2005a]	Second International Workshop on Software Product Line Testing. *SPLiT Proceedings and Presentations*. http://www.biglever.com/split2005/presentations.html (2005).
[STARS 1996a]	Software Technology for Adaptable Reliable Systems (STARS). *Organization Domain Modeling (ODM) Guidebook Version 2.0* (STARS-VC-A025/001/00). Manassas, VA: Lockheed Martin Tactical Defense Systems, 1996.
[Sun 2007a]	Sun Microsystems, Inc. *Sun Microsystems Home Page*. http://www.sun.com/ (2007).
[Svahnberg 2000a]	Svahnberg, M. & Bosch, J. "Issues Concerning Variability in Software Product Lines," 50-60. *Proceedings of the Third International Workshop on Software Architectures for Product Families*. Las Palmas de Gran Canaria, Spain, March 15-17, 2000. Berlin, Germany: Springer, 2000.
[Szyperski 2002a]	Szyperski, C. *Component Software: Beyond Object-Oriented Programming, 2nd ed.*. Boston, MA: Addison-Wesley, 2002.
[Thiel 2000a]	Thiel, S. & Peruzzi, F. "Starting a Product Line Approach for an Envisioned Market: Research and Experience in an Industrial Environment," 495-512. *Software Product Lines: Proceedings of the First Software Product Line Conference (SPLC1)*. Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic Publishers, 2000.
[Tichy 1989a]	Ticky, N. & Charan, R. "Speed, Simplicity, Self-Confidence: An Interview with Jack Welch." *Harvard Business Review 67*, 5 (September-October 1989): 112-120.
[Tilley 1997a]	Tilley, S. R. *Discovering DISCOVER* (CMU/SEI-97-TR-012, ADA331014). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997.
[Tilley 1998a]	Tilley, S. R. *A Reverse-Engineering Environment Framework* (CMU/SEI-98-TR-005, ADA343688). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998.

[Toft 2000a]	Toft, P.; Coleman, D.; & Ohta, J. "A Cooperative Model for Cross-Divisional Product Development for a Software Product Line," 111-132. *Software Product Lines: Proceedings of the First Software Product Line Conference (SPLC1)*. Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic Publishers, 2000.
[Tolvanen 2005a]	Tolvanen, Juha-Pekka & Kelly, Steven. "Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences," 198-209. *Proceedings of the Ninth International Software Produce Lines Conference (SPLC 2005)*. Rennes, France, September 26-29, 2005. New York, NY: Springer, 2005.
[Tompkins 2004a]	Tompkins, J. *40 Risks to Establishing an Outsourcing Relationship*. http://www.tompkinsinc.com/operations/info/40_Outsourcing_Risks.pdf (2004).
[Tracz 1995a]	Tracz, W. *Confessions of a Used Program Salesman: Institutionalizing Software Reuse*. New York, NY: Addison-Wesley, 1995.
[Tracz 1988a]	Tracz, W. "RMISE Workshop on Software Reuse Meeting Summary," 41-53. *Software Reuse: Emerging Technology*. Washington, DC: Computer Society Press, 1988.
[TreeAge 1999a]	TreeAge Software, Inc. DATA Interactive White Paper. http://server.treeage.com/objdocs/start/whitepaper.php3 (1999).
[Ulrich 2002a]	Ulrich, W. M. *Legacy Systems: Transformation Strategies*. Upper Saddle River, NJ: Prentice Hall, 2002 (ISBN 0-13-044927-X).
[Van Zyl 2000a]	Van Zyl, J. & Walker, A. J. "Strategic Product Development: A Strategic Approach to Taking Software Products to Market Successfully," 86-111. *Software Product Lines: Proceedings of the First Software Product Line Conference (SPLC1)*. Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic Publishers, 2000.
[Vollman 1994a]	Vollman, T. "Standards Support for Software Tool Quality Assessment," 29-39. *Proceedings of the Third Symposium on Assessment of Quality Software Development Tools*. Washington, DC, June 7-9, 1994. Los Alamitos, CA: IEEE Computer Society Press, 1994.
[Vu 2000a]	Vu, J. "Findings of the Managing Software Innovation and Technology Change Workshop: Managing Technology Transfer" [CD-ROM]. *Proceedings of the Software Engineering Process Group (SEPG) 2000*. Seattle, WA, March 20-23, 2000. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000.
[Wallnau 1997a]	Wallnau, K.; Weiderman, N.; & Northrop, L. *Distributed Object Technology with CORBA and Java: Key Concepts and Implications* (CMU/SEI-97-TR-004, ADA327035). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997.
[Wallnau 2002a]	Wallanu, Kurt; Hissam, Scott A.; & Seacord, Robert C. *Building Systems from Commercial Components*. Upper Saddle River, NJ: Addison-Wesley,

	2002.
[Walrad 2002a]	Walrad, Chuck & Strom, Darrel. "The Importance of Branching Models in SCM." *IEEE Computer 35*, 9 (September 2002): 31-38.
[Wappler 2000a]	Wappler, T. "Remember the Basics: Key Success Factors for Launching and Institutionalizing a Software Product Line," 73-84. *Software Product Lines: Proceedings of the First Software Product Line Conference (SPLC1)*. Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic Publishers, 2000.
[Wartik 1992a]	Wartik, S. & Prieto-Diaz, R. "Criteria for Comparing Reuse-Oriented Domain Analysis Approaches." *International Journal of Software Engineering and Knowledge Engineering 2*, 3 (September 1992): 403-431.
[Weiderman 1997a]	Weiderman, N.; Northrop, L.; Smith, D.; Tilley, S.; & Wallnau, K. *Implications of Distributed Object Technology for Reengineering* (CMU/SEI-97-TR-005, ADA326945). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997.
[Weiss 1999a]	Weiss, D. M. & Lai, C. T. R. *Software Product-Line Engineering: A Family-Based Software Development Process*. Reading, MA: Addison-Wesley, 1999.
[Westfechtel 2003a]	Westfechtel, Bernhard & Conradi, Reidar. "Software Architecture and Software Configuration Management," 24-39. *Proceedings of the ICSE Workshops SCM 2001 and SCM 2003: Selected Papers*. van der Hoek, A. & Westfechtel, B., eds. Lecture Notes in Computer Science Volume 2649. Berlin, Germany: Springer-Verlag, 2003.
[Wheeler 2005a]	Wheeler, David A. *Why Open Source Software/Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!* http://www.dwheeler.com/oss_fs_why.html (2005).
[Wijnstra 2000a]	Wijnstra J. "Supporting Diversity with Component Frameworks as Architectural Elements," 50-59. *Proceedings of the 22nd International Conference on Software Engineering (ICSE)*. Limerick, Ireland, June 4-11, 2000. New York, NY: ACM, 2000.
[Wikipedia 2007a]	Wikipedia. *Agile Software Development*. http://en.wikipedia.org/wiki/Agile_software_development (2007).
[Williams 1999a]	Williams, R.; Pandelios, G.; & Behrens, S. *Software Risk Evaluation (SRE) Method Description (Version 2.0) & SRE Team Members Notebook (Version 2.0)* (CMU/SEI-99-TR-029, ADA001008). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.
[Wingerd 2005a]	Wingerd, Laura & Seiwald, Christopher. *High-Level Best Practices in Software Configuration Management*. http://www.perforce.com/perforce/bestpractices.html (2005).
[Withey 1996a]	Withey, J. *Investment Analysis of Software Assets for Product Lines* (CMU/SEI-96-TR-010, ADA315653). Pittsburgh, PA: Software

	Engineering Institute, Carnegie Mellon University, 1996.
[Woods 1999a]	Woods, S. G.; Carriere, S. J.; & Kazman, R. "A Semantic Foundation for Architectural Reengineering," 391-398. *Proceedings of the ICSM99*. Oxford, UK, August 30 - September 3, 1999. Oxford, UK: Oxford Press, 1999.
[Yacoub 2000a]	Yacoub, S.; Mili, A.; Kaveri, C.; & Dehlin, M. "A Hierarchy of COTS Certification Criteria," 397-412. *Software Product Lines: Proceedings of the First Software Product Line Conference (SPLC1)*. Denver, Colorado, August 28-31, 2000. Boston, MA: Kluwer Academic Publishers, 2000.
[Zahran 1998a]	Zahran, Sami. *Software Process Improvement : Practical Guidelines for Business Success*. Reading, MA: Addison-Wesley, 1998 (ISBN 0-201-17782-X).
[Zhang 2005a]	Zhang, Weishan & Jarzabek, Stan. "Reuse Without Compromising Performance: Industrial Experience from RPG Software Product Line for Mobile Devices," 57-69. *Proceedings of Software Product Lines: Ninth International Conference*. Rennes, France, September 26-29, 2005. New York, NY: Springer, 2005.
[Zhao 1999a]	Zhao, J. "Bibliography on Software Architecture Analysis." *Software Engineering Notes 24*, 3 (May 1999): 61-62.
[Zona 1999a]	Zona Research, Inc. *Enterprise JavaBeans Technology, a Business Benefits Analysis*. (1999). http://java.sun.com/products/ejb/pdf/zona.pdf
[Zubrow 2003a]	Zubrow, Dave & Chastek, Gary. *Measures for Software Product Lines* (CMU/SEI-2003-TN-031). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.