Introduction to the SEI's Software Product Line Framework®

http://www.sei.cmu.edu/productlines/framework.html#outline

Instructor: Peter Abowd MSE

Goals of the Framework

- Software product lines similar to traditional Product lines
  - Commonize parts for less complex manufacturing
  - **Goals :**
    - **Identify the foundational concepts underlying software product lines and the essential activities to consider before developing a product line**
    - **Identify practice areas that an organization developing software product lines must master**
      - Developing Product lines imposes additional needs to common pratices
    - **Define practices in each practice area**
      - For example, "Configuration Management" is a practice area that applies to any software development effort, but it has special implications for product line development. Thus, we identify "Configuration Management" as a practice area, but we also are able to define one or more effective configuration management practices for product lines.
  - **Provide guidance to an organization about how to move to a product line approach for software**

Understanding Your Needs

- The type of system(s) being built
- The depth of domain experience
- The legacy assets on hand
- The organizational goals
- The maturity of artifacts and processes
- The skill level of the personnel available
- The production strategy embraced

Software Product Lines

- **A *software product line* is**
  - ***"a set of software-intensive systems that share a common, managed set of features satisfying 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."***
- Substantial production economies can be achieved
  - The systems in a software product line are developed from a common set of assets in a prescribed way
  - They are not developed separately, from scratch or in an arbitrary fashion.
  - It is exactly these production economies that make the software product line approach attractive.

Software Economy of Scale

- How is production made more economical?
  - 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
    - Assembling the collection according to the rules of a common, product-line-wide architecture
  - Building a new product (system) becomes more a matter of assembly or generation than one of creation;
    - The predominant activity is integration rather than programming.
  - For each software product line there is a predefined guide or plan that specifies the exact product-building approach.

Software Product Lines

- The common set of assets and the plan for how they are used to build products
  - Don't just materialize without planning
  - They require investment
- They require organizational foresight
  - Investment, planning, and direction
- They require strategic thinking that looks beyond a single product
  - The disciplined use of the common assets to build products doesn't just happen
    - Management must direct, track, and enforce the use of the assets.
- Software product lines are as much about business practices as they are about technical practices.
- Software product lines give *economies of scope*,
  - Means that organizations take economic advantage of the fact that many of thier products are very similar—not by accident, but because they planned it that way
  - Organizations make deliberate, strategic decisions and are systematic in effecting those decisions

Reuse Vs. Product Lines -1

- Historic Reuse
  - Past reuse agendas have focused on the reuse of relatively small pieces of code—that is, small-grained reuse.
  - Organizations have built reuse libraries containing algorithms, modules, objects, or components.
  - Almost anything a software developer writes goes into the library. Other developers are then urged (and sometimes required) to use what the library provides instead of creating their own versions.
  - Challenges with the common reuse model
    - It often takes longer to locate these small pieces and integrate them into a system than it would take to build them anew.
    - Documentation, if it exists at all, might explain the situation for which the piece was created but not how it can be generalized or adapted to other situations.
    - The benefits of small-grained reuse depend upon
      - Predisposition of the software engineer to use what is in the library
      - Suitability of what is in the library for the engineer's particular needs
      - Successful adaptation and integration of the library units into the rest of the system
    - If reuse occurs under these conditions it is fortuitous with low payoff
  - In a software product line approach, the reuse is planned, enabled, and enforced—the opposite of opportunistic

Reuse Vs. Product Lines -2

- Traditional Reuse attempts often falter in to a clone and own
  - The “reusing” product replicates the code base and becomes its owner
    - Example if ineffective configuration management

Vs

- Planned Reuse of a Product Line
  - Software product lines reuse assets that were designed explicitly for reuse.
  - The product line is treated as a whole
    - Not as multiple products that are viewed and maintained separately
  - In mature product line organizations
    - The concept of multiple products disappears
    - Each product is simply a tailoring of the common assets
      - The assets constitute the core of each product
      - A product may also contine a small collection of additional unique artifacts
    - It is the core assets that are designed carefully and evolved over time
    - It is the core assets that are the organization's premiere intellectual property

Reuse Vs. Product Lines -3

- Building a Product from a Product Line
  - Components are assembled in a prescribed way
  - Includes exercising built-in variability mechanisms in the components to put them to use in specific products
  - The prescription comes from both the architecture and the production plan
  - These prescritpions and plans are missing from standard component-based development

Reuse Vs. Product Lines -4

- Releases and Versions Vs P-Lines
  - Organizations routinely produce new releases and versions of products
  - Each of these new versions and releases is typically constructed using
    - the architecture, components, test plans, and other features of the prior releases.
  - Why are software product lines different?
    - In a product line there are multiple simultaneous products, all of which are going through their own cycles of release and versioning simultaneously
    - The evolution of a single product must be considered within a broader context—namely, the evolution of the product line as a whole
    - In a product line, an early version of a product that is still considered to have market potential can easily be kept as a viable member of the family
      - It is an instantiation of the core assets, just like other versions of other products

Costs and Benefits of ProdUct Lines

- Large-scale productivity gains
- Decreased time-to-market
- Increased product quality
- Increased customer satisfaction
- More efficient use of human resources
- Ability to effect mass customization
- Ability to maintain market presence
- Ability to sustain unprecedented growth

Benefits in Detail

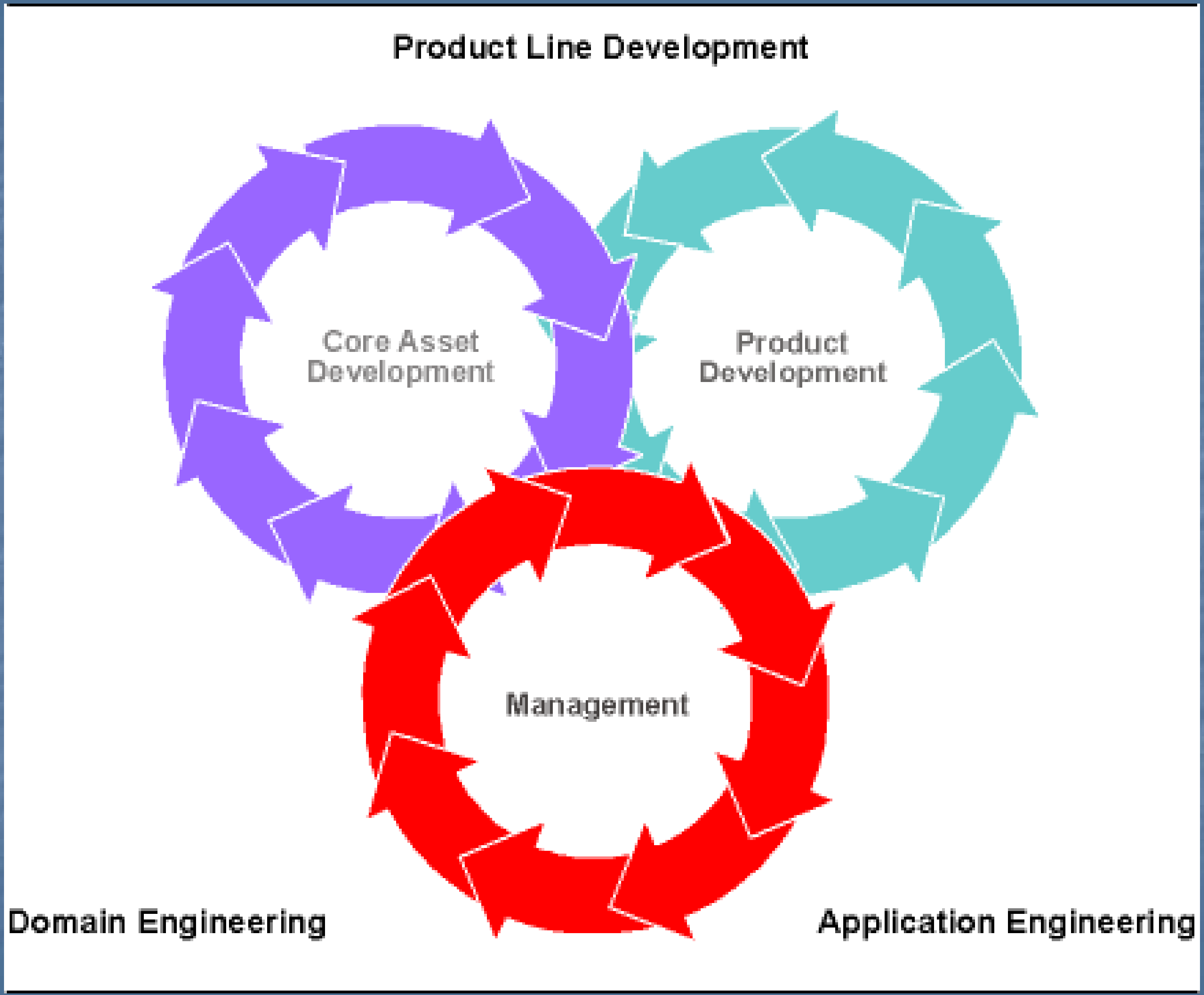
- **Requirements:** There are common product line requirements. Product requirements are deltas to this established requirements base. Extensive requirements analysis is saved. Feasibility is assured.
- **Architecture:** An architecture for a software system represents a large investment of time from 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 but any 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 that 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. They 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.

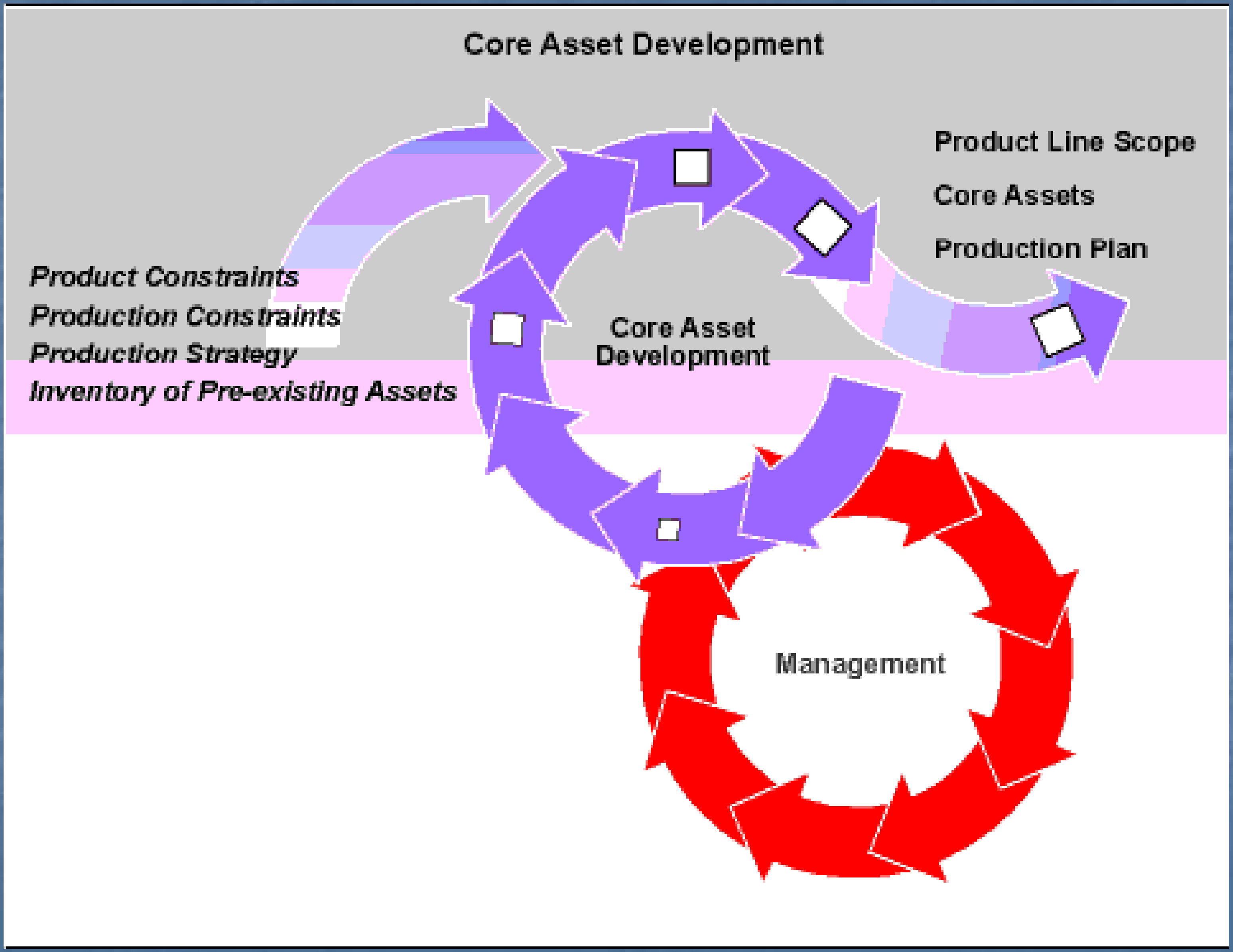
Costs and Benefits of Product Lines		
Core Asset	**Benefit**	**Additional Cost**
Requirements: The requirements are written for the group of systems as a whole, with requirements for individual systems specified by a delta or an increment to the generic set.	Commonality and variation are documented explicitly, 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 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 of processes among processors, on creation of new processes, or on synchronization of existing processes.

Costs and Benefits of Product Lines		
Core Asset	**Benefit**	**Additional Cost**
Business case, market analysis, marketing collateral, cost and schedule estimates: These are the up-front business necessities involved in any product. Generic versions are built that support the entire product line.	All of the business and management artifacts involved in turning out already exist at least in a generic form and can be reused.	All of these artifacts must be generic, or be made extensible to accommodate product variations.
Tools and processes for software development and for 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 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 for the differences between managing a product line and managing a single product.
Test cases, test plans, 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 of the testing artifacts must be more robust because they will support more than one product. They also must be extensible to accommodate variation among the products.
People, skills, 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. Their 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.

Key Terms

- A *software product line* is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.
- *Core assets* are those reusable artifacts and resources that form the basis for the software product line. Core assets often include, but are not limited to, the architecture, reusable software components, domain models, requirements statements, documentation and specifications, performance models, schedules, budgets, test plans, test cases, work plans, and process descriptions.
- *Development* is a generic term used to describe how core assets (or products) come to fruition. Software enters an organization in any one of three ways: the organization can build it itself, purchase it, or commission it
- A *domain* is a specialized body of knowledge, an area of expertise, or a collection of related functionality. For example, the telecommunications domain is a set of telecommunications functionality, which in turn consists of other domains such as switching, protocols, telephony, and network. A telecommunications software product line is a specific set of software systems that provide some of that functionality.
- *Software product line practice* is the systematic use of core assets to assemble, instantiate, or generate the multiple products that constitute a software product line. The choice of verb depends on the production approach for the product line. Software product line practice involves strategic, large-grained reuse.


Product Line Development
Core Asset Development
Product Development
Management
Domain Engineering
Application Engineering


Core Asset Development
Product Line Scope
Core Assets
Production Plan
Product Constraints
Production Constraints
Production Strategy
Inventory of Pre-existing Assets
Core Asset Development
Management

Core Asset Development

- Three things are required for a production capability to develop products
  - These three things are the outputs of the Core Asset development activity
    - **Product line scope**
    - **Core assets**
    - **Production plan**

Product Line Scope

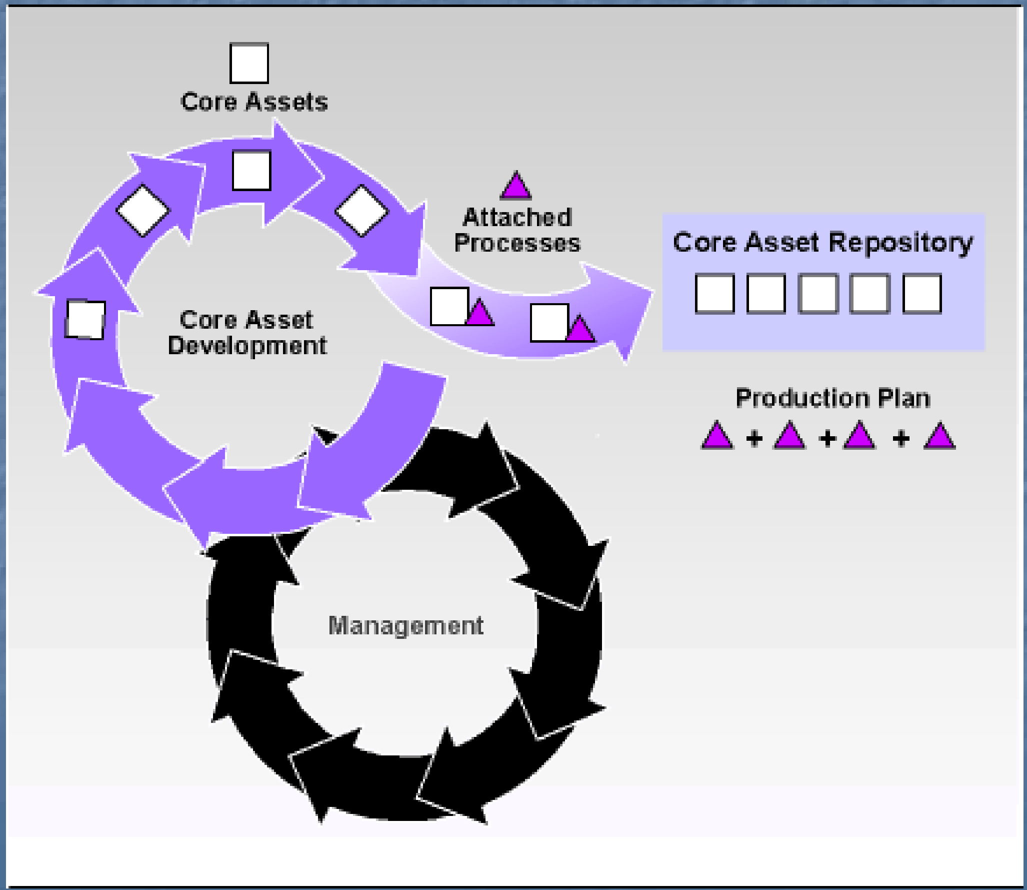
- The product line scope is a description of the products that will constitute the product line or that the product line is capable of including.
- Successful Product Line must be scoped carefully
  - Too Small
    - Not enough future proofing of core assets
  - Too Large
    - Difficult to manage complexity in Core

Core Asset Development

- Architecture
  - Covers the general product line and defines the variation points
- Software Components
- Knowledge of relevant patterns and frameworks
  - Available inventory of preexisting assets

Core Asset Attached Process

- Use the product line requirements as the baseline requirements
- Specify the variation requirement for any allowed variation point
- Add any requirements outside the set of specified product line requirements
- Validate that the variations and extensions can be supported by the architecture.



Production Plan

- A production plan prescribes how the products are produced from the core assets
  - Set of these attached processes with the necessary glue
  - Each product in the product line will vary consistent with predefined variation points
  - The production plan should describe how specific tools are to be applied in order to use, tailor, and evolve the core assets.
    - Measurement plan too
- As will be seen in Product Development, these three outputs are necessary ingredients for feeding the product development activity, which turns out products that serve a particular customer or market niche.
  - Product Line Scope
  - Core Assets
  - Production Plan

Inputs to Core Asset Development

Product constraints

The core assets must capitalize on the commonalities and accommodate envisioned variation with minimal tradeoff to product quality drivers such as security, reliability, usability, and so on

Production constraints

i.e. whether to invest in a generator environment or rely on manual coding. This in turn will drive decisions about what kind of variability mechanisms to provide in the core assets, and what form the overall production plan will take

Production strategy

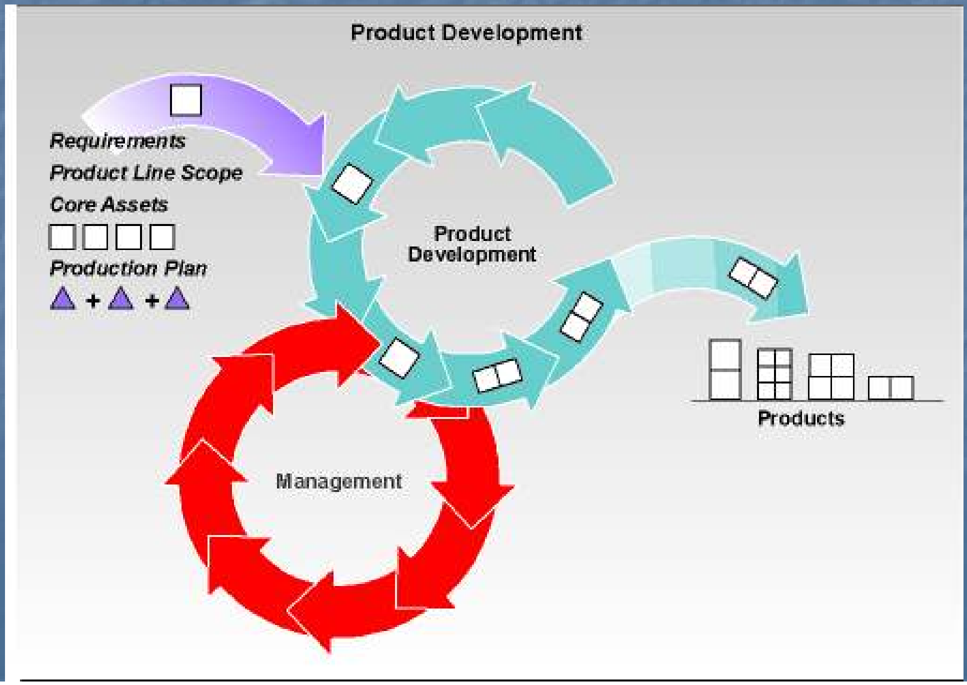
The production strategy is the overall approach for realizing the core assets and products. Will the product line be built proactively , reactively, or using some combination

Inventory of preexisting assets

Legacy systems embody an organization's domain expertise and/or define its market presence. The product line architecture, or at least pieces of it, may borrow heavily from proven structures of related legacy systems.

Product Development Inputs

- Requirements for a particular product
  - Often expressed as a delta or variation from some generic product description contained in the product line scope or as a delta from the set of product line requirements
- The product line scope
  - Indicates whether or not the product under consideration can be feasibly included in the product line
- The core assets from which the product is built
- The production plan
  - Details how the core assets are to be used to build the product



Management

- Technical management
  - oversees the core asset development and to the product development activities by ensuring that the groups who build core assets and the groups who build products are engaged in the required activities, follow the processes defined for the product line, and collect data sufficient to track progress
- Organizational management
  - determines a funding model that will ensure the evolution of the core assets and then provides the funds accordingly. Organizational management also orchestrates the technical activities in and iterations between the essential activities of core asset development and product development.
- Product Line Management
  - This person must be a strong, visionary leader who can keep the organization squarely pointed toward the product line goals, especially when the going gets rough in the early stages. Leadership is required for software product line success. <u>Management and leadership are not always synonymous</u>.

3 Activities of Product Lines

- **Core Asset Development**
- **Product Development**
- **Managment**
- **Many organizations begin a software product line by developing the core assets first.**
  - **This is referred to as a *proactive* approach.**
  - **They define their product line scope to define the set of systems that will constitute their product line. This scope definition provides a kind of mission statement for designing the product line architecture, components, and other core assets with the right built-in variation points to cover the scope.**
  - **Producing any product within that scope becomes a matter of exercising the variation points of the components and architecture—that is, configuring—and then assembling and testing the system.**
- **Other organizations begin with one or a small number of products they already have and from these generate the product line core assets and future products.**
  - **This is referred to as a *reactive* approach.**
- Iterations are likely and useful between these 2 approaches

End of Lecture 1

Practice Areas

- A practice area is a body of work or a collection of activities that an organization must master to successfully carry out the essential work of a product line.
- 3 Categories
  - Software Engineering
  - Technical Management
  - Organizational Management

Software Engineering Practice Areas

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

We will Discuss a Subset of these practices

Architecture Definition

Software Engineering Practice Area

- *The software architecture of a program or computing system is the structure or structures of the system,*
  - *which comprise software elements,*
  - *the externally visible properties of those elements,*
  - *and the relationships among them. "*
  - *Externally visible" properties, we are referring to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on [Bass et al]*

- By making "externally visible properties" of elements part of the definition, we intentionally and explicitly include elements' interfaces and behaviors as part of the architecture
- By contrast, design decisions or implementation choices that do not have system-wide ramifications or visibility are not architectural
- The architecture is the place where understanding begins

Architectural Definition -2

The complete architecure is composed these:

- Architectural Requirements
  - An architecture's constraints go far beyond implementing the required *behavior*
    - Quality Attributes
    - System level interactions
    - Business Goals
    - Sources for the Components
      - Internal, External, 3rd Party Contracted
- Component Interface Definition
- Connecting Components (i.e. middleware)
- Architectural Documentation and Views

Architectural Views

- A view is a representation of a set of system elements and the relationships among them

- A view can be thought of as a projection of the architecture that includes certain kinds of information and suppresses other kinds

- There are many views of an architecture
  - Choosing which ones to document is a matter of what information you wish to convey

Product Line Unique Architectural Aspects

- Concerned with identifying and providing mechanisms to achieve a set of explicitly allowed variations
  - When variations are exercised these become products
  - A conventional architecture almost any instance will do as long as the (single) system's behavioral and quality goals are met
- Architecture Must Handle Complexity
  - In a conventional architecture, the mechanism for achieving different instances almost always comes down to modifying the code
  - In a software product line, support for variation can take many forms
- Integration may assume a greater role for software product lines than for one-off systems simply because of the number of times it's performed
- There must be documentation for the product line architecture as it resides in the core asset base and for each product's architecture

Specific Architecture Definition Practices

- Architecture Definition
- Attribute Driven Design
- Architectural Patterns
- Quality Attribute Workshop
- Aspect Oriented Programming
- Many of these covered in courses already

Important Practice: Product Builders Guide

- **Introduction:** goals and purpose of the document; intended audience; basic common assumptions; applicable development standards
- **Sources of other information:** references to documents containing the product line architecture definition (which is maintained separately from the product builder's guide because its stakeholders include more than product builders) and associated information such as terms and terminology, the architecture's goals, architecture training materials, development standards, and configuration management procedures and policies
- **Basic concepts:** What is a variation point? What mechanisms for realizing variation points have been used in this architecture? What is the relation between the product line architecture and the architecture for a particular product? What is an architecture layer, and how is the concept used? What is a service (in this case, the basic unit of reuse provided by the architecture)? And so forth.
- **Service component catalogue:** This organization's product line architecture contains some preintegrated units of functionality called service components that product builders can use to construct products. This section catalogues those service components, defines their interfaces, and explains how service components related to each other.
- **Building an application:** This section gives code templates and examples for building applications. It progresses incrementally. First, how do you build the most trivial application possible, one that perhaps does nothing but start a process running? Then, how do you build the most trivial application that actually does something observable, the domain's equivalent of the ubiquitous "Hello, world!" program that was the first computer program many of us ever wrote? Then, how do you build an application that contains the functions common to many of the products in the product line? Then, how do you build an application that runs on a single processor? Distributed across multiple processors? And so forth. The examples show how to instantiate the architecture's variation points at each step along the way.
- **Performance engineering:** This section presented guidelines on how to build a product when performance was a concern.

Achieving Product Variability

- **Inheritance:** in object-oriented systems, used when a method needs to be implemented differently (or perhaps extended) for each product in the product line
- **Extensions and extension points:** used when parts of a component can be augmented with additional behavior or functionality
- **Parameterization:** used when a component's behavior can be characterized abstractly by a placeholder that is then defined at build time. Macros and templates are forms of parameterization.
- **Configuration and module interconnection languages:** used to define the build-time structure of a system, including selecting (or deselecting) whole components
- **Generation:** used when there is a higher-level language that can be used to define a component's desired properties
- **Compile-time selection of different implementations:** The variable *#ifdefs* can be used when variability in a component can be realized by choosing different implementations.

Architectural Definition Practice Risks

- **Lack of a skilled architect:** A product line architect must be skilled in current and promising technologies, the nuances of the application domains at hand, modern design techniques and tool support, and professional practices such as the use of architectural patterns. The architect must know all of the sources of requirements and constraints on the architecture, including those (such as organizational goals) not traditionally specified in a requirements specification
- **Lack of sound input:** The product line scope and production strategy must be well defined and stable. The requirements for products must be articulated clearly and completely enough so that architectural decisions may be reliably based on them.
- **Poor communication:** The best architecture is useless if it is documented and communicated in ways that its consumers-the product builders-cannot understand. An architecture whose documentation is chronically out of date is effectively the same as an undocumented architecture. There must be clear and open two-way communication channels between the architect and the organizations using the architecture.
- **Lack of supportive management and culture:** There must be management support for the creation and use of the product line architecture, especially if the architecture group is separate from the product development group. Failing this, product groups may "go renegade" and make unilateral changes to the architecture, or decline to use it at all, when turning out their systems. There are additional risks if management does not support the strong integration of system and software engineering.
- **Architecture in a vacuum:** The exploration and definition of software architecture cannot take place in a vacuum separate from system architecture.
- **Poor tools:** There are precious few tools for this practice area, especially those that help with designing, specifying, or exercising an architecture's variability mechanisms–a fundamental part of a product line architecture.
- **Poor timing:** Declaring an architecture ready for production too early leads to stagnation, while declaring it too late may allow unwanted variation. Discretion is needed when deciding when and how firmly to freeze the architecture. The time required to fully develop the architecture also may be too long. If product development is curtailed while the product line architecture is being completed, developers may lose patience, management may lose resolve, and salespeople may lose market share.

Architectural Definition Practice Risks -2

- **Inappropriate parameterization:** Overparameterization can make a system unwieldy and difficult to understand. Underparameterization can eliminate some of the necessary customizations of the system. The early binding of parameters can also preclude easy customization, while the late binding of parameters can lead to inefficiencies.
- **Inadequate specifications:** Components may not integrate properly if their specifications are sketchy or limited to static descriptions of individual services.
- **Decomposition flaws:** A component may not provide the functionality needed to implement the system correctly if there is not an appropriate decomposition of the required system functionality.
- **Wrong level of specificity:** A component may not be reusable if the component is too specific or too general. If the component is made so general that it encompasses multiple domain concepts, the component may require complex configuration information to make it fit a specific situation and therefore be inherently difficult to reuse. The excessive generality may also tax performance and other quality attributes to an unacceptable point. If the component is too specific, there will be few situations in which it is the correct choice.
- **Excessive intercomponent dependencies:** A component may become less reusable if it has excessive dependencies on other components.

Architectural Evaluation Practices

Software Engineering Practice Area

- The evaluation can be done at a variety of stages during design
  - For example, the evaluation can occur when the architecture is still on the drawing board and candidate structures are being weighed.
  - The evaluation can also be done later, after preliminary architectural decisions have been made, but before detailed design has begun.
  - The evaluation can even be done after the entire system has been built
- The outputs will depend on the stage at which the evaluation is performed
- Enough design decisions must have been made so that the achievement of the requirements and quality-attribute goals can be analyzed
- The more architectural decisions that have been made, the more precise the evaluation can be
  - However, the more decisions that have been made, the more difficult it is to change them

PLA Unique Impact on Evaluation

- Architecture assumes a dual role.
- There is the architecture for the product line as a whole
- There are architectures for each of the products
  - The latter are produced from the former by exercising the built-in variation mechanisms to achieve instances
- Both should be evaluated
- The evaluation should focus upon the variations
  - Make sure they are appropriate
  - They offer sufficient flexibility to cover the product line's intended scope
  - They can be exercised in a way that lets products be built quickly
  - They do not impose unacceptable runtime performance costs

Specific Architectural Evaluation Practices

- **ATAM^{SM}**: The Architecture Tradeoff Analysis MethodSM (ATAM) is a scenario-based architecture evaluation method that focuses on a system's quality goals
  - ATAM can be used to evaluate both product line and product architectures at various stages of development
- Software performance engineering (SPE)
  - Method for making sure that a design will allow a system to meet its performance goals before it has been built
- Active Reviews for Intermediate Designs (ARID)
  - A hybrid design review method that combines the active design review philosophy of ADRs with the scenario-based analysis of the ATAM and SAAM. ARID was created to evaluate partial (subsystem, for example) designs in their early or conceptual phases, before they are fully documented
- An Active Design Review (ADR)
  - A technique that can be used to evaluate an architecture still under construction

Architectural Evaluation Practice Risks

- The major risk associated with this practice is failing to perform an effective architecture evaluation that will prevent unsuitable architectures from being allowed to pollute a software product line effort
  - **Wrong people involved in the evaluation:** If the architect is not involved in the evaluation, it is unlikely that enough information will be uncovered to make the evaluation worthwhile. Similarly, if the architecture's stakeholders are not involved, the comprehensive goals and requirements for the architecture (against which it must be evaluated) will not emerge.
  - **Wrong time in the life cycle:** If the review is too early, not enough decisions have been made, so there isn't anything to evaluate. If the review is too late, little can be changed as a result of the evaluation.
  - **No time for evaluation:** If time is not planned for the evaluation, the people who need to be involved will not be able to give it their attention, the evaluation will not be conducted effectively, and the results will be superficial at best.
  - **Wrong interpretation of evaluation:** The results of any architecture evaluation should not be seen as a complete enumeration of all of the risks in the development. Process deficiencies, resource inadequacies, personnel issues, and downstream implementation problems are all risks unlikely to be exposed by an architecture evaluation.
  - **Failure to reevaluate:** As the architecture inevitably evolves, or the criteria for its suitability inevitably evolve, it should be reevaluated (perhaps using a lightweight version of the original evaluation) periodically to give the organization confidence that they are on the right track.

Component Development Practices

Software Engineering Practice Area

- *A software component is a unit of 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.*
- Component development
  - The production of components that implement specific functionality within the context of a software architecture
  - The functionality is encapsulated and packaged, then integrated with other components using an interconnection method
- Component-based software development shifts the emphasis from *programming software to composing software systems*
- <u>Product line development is the creation of component which handle the specified variability of the product line</u>

Application to Product Development

- Components for a product are:
  - Used directly from the core asset base
  - Used directly after binding the built-in variabilities
  - Used after modification or adaptation
    - Wrapping or modifying
  - Developed anew
    - Only after Exhaustive search
    - Should determine if it then becomes a core asset

Component Level Variability Practices

Types of Variation [Jacobson 97]		
Mechanism	**Time of Specialization**	**Type of Variability**
Inheritance	At class definition time	Specialization is done by modifying or adding to existing definitions. Example: LongDistanceCall inherits from PhoneCall.
Extension	At requirements time	One use of a system can be defined by adding to the definition of another use. Example: WithdrawalTransaction extends BasicTransaction.
Uses	At requirements time	One use of a system can be defined by including the functionality of another use. Example: WithdrawalTransaction uses the Authentication use.
Configuration	Previous to runtime	A separate resource, such as file, is used to specialize the component. Example: JavaBeans properties file
Parameters	At component implementation time	A functional definition is written in terms of unbound elements that are supplied when actual use is made of the definition. Example: calculatePriority(Rule)
Template instantiation	At component implementation time	A type specification is written in terms of unbound elements that are supplied when actual use is made of the specification. Example: ExceptionHandler<Container>
Generation	Before or during runtime	A tool that produces definitions from user input. Example: Configuration wizard

Component Development Practice Risks

- **Not enough variability:**
  - Components not only must meet their behavioral and quality requirements but also must be tailorable in preplanned ways to enable product developers to instantiate them quickly and reliably in the correct forms for specific products
- **Too much variability:**
  - Building in too much variability can prevent the components from being understood well enough to be used effectively, or can cause unforeseen errors when the variabilities conflict with each other.
- **Choosing the wrong variation mechanism(s) for the job:**
  - The wrong choice can result in components that cannot be tailored at the time they need to be.
- **Poor quality of components:**
  - Components of poor quality will set back any effort, but poor core asset components will undermine the entire product line. Product builders will lose confidence with the core asset builders, and pressure to bypass them will mount. The "Testing" practice area should be applied to ameliorate this risk.

Mining Existing Assets

Software Engineering Practice Area

- Mining existing assets refers to resurrecting and rehabilitating a piece of an old system to serve in a new system for which it was not originally intended
  - Not Just Code artifacts
  - Business models
  - rule bases, requirements specifications, schedules, budgets, test plans, test cases, coding standards, algorithms, process definitions, performance models
  - If good documentation does not exist, the process of architecture reconstruction may need to be employed
    - Reconstruction will reveal the interactions and relations among the architecture's components
- Focus first on large-grained assets that can be wrapped or that will require only interface changes rather than changes in large chunks of the underlying algorithms
  - performance, modifiability, reliability, and other nonbehavioral qualities

Product Line Aspects of Asset Mining

- Mined assets
  - Must be (re)packaged with reuse in mind
  - Must meet the product line requirements
  - Must align with the product line architecture
  - Must meet the quality goals consistent with the goals of the product line
- Product line systems emphasize quality attributes such as *maintainability* and *suitability*
  - These attributes become more important over time
  - Mined assets for product lines that are suboptimal in fulfilling specific tasks may still be valuble if they meet the critical quality-attribute goals

Asset Mining Application to Core Asset Development

- Key activity of Core Asset Development
- Candidate software assets must
  - Align with the product line architecture
  - Meet specified component behavior requirements
  - Accommodate any specified variation points

Specific Practices for Mining Core Assets

Options Analysis for Reengineering (OAR): OAR is a method that can be used to evaluate the feasibility and economy of mining existing components for a product line.

- **Establish mining context:** First, capture your organization's product line approach, legacy base, and expectations for mining components. Establish the programmatic and technical drivers for the effort, catalogue the documentation available from the legacy systems, and identify a broad set of candidate components for mining. This task establishes the needs of the mining effort and begins to illuminate the types of assets that will be most relevant for mining. It also identifies the documentation and artifacts that are available, and it enables focused efforts to close gaps in existing documentation.
- **Inventory components:** Next, identify the legacy system components that can potentially be mined for use in a product line core asset base. During this activity, identify required characteristics of the components (such as functionality, language, infrastructure support, and interfaces) in the context of the product line architecture. This activity creates an inventory of candidate legacy components together with a list of the relevant characteristics of those components. It also creates a list of those needs that cannot be satisfied through the mining effort.
- **Analyze candidate components:** Next, analyze the candidate set of legacy components in more detail to evaluate their potential for use as product line components. Screen them on the basis of how well they match the required characteristics. This activity provides a list of candidate components, together with estimates of the cost and effort required for rehabilitating those components.
- **Analyze mining options:** Next, analyze the feasibility and viability of mining various aggregations of components on the basis of cost, effort, and risk. Assemble different aggregations of components and weigh their costs, benefits, and risks.
- **Select mining option:** Finally, select the mining option that can best satisfy the organization's mining goals by balancing the programmatic and technical considerations. First, establish drivers for making a final decision, such as cost, schedule, risks and difficulty. Tradeoffs often can be established by this activity. Evaluate each mining option (component aggregation) on the basis of how well it satisfies the most critical driver. Select an option, and then develop a final report to communicate the results.

Specific Practices for Mining Core Assets

- **Architecture recovery/reconstruction tools:**
  - Some tools that are available to assist in the architecture reconstruction process include Rigi [Muller 88], the Software Bookshelf [Finnegan 97], DISCOVER [Tilley 98], and the Dali workbench [Kazman 98] and the ARMIN tool [O'Brien 03].
- **Mining Architectures:**
  - In some cases the software architecture of an existing system can become the product line architecture.
  - Mining Architectures for Product Lines (MAP) is a method that determines whether the architectures of existing systems are similar and whether the corresponding systems have the potential of becoming a software product line [O'Brien 01].
- **Requirements Reuse and Feature Interaction Management**
  - understanding of interaction management is key to understanding how to reuse requirements and describes a conceptual process framework for formulating and reusing requirements
- **Wrapping:**
  - Wrapping involves changing the interface of a component to comply with a new architecture, but not making other changes in the component's internals.
- **Adapting components:**
  - Software components that are being used in a context other than the one for which they were originally developed often do not exactly fit their assigned roles.

Mining Assets Practice Risks
(Searching Risks)

- **Flawed search:**
  - The search for reusable assets may be fruitless, resulting in a waste of time and resources. Or, relevant assets may be overlooked, resulting in time and resources being wasted duplication of what already exists. A special case of the latter is when noncode assets are shortsightedly ignored. To minimize both of these risks, build a catalogue of your reusable assets (including noncode assets) and treat that catalogue as a core asset of the product line. It will save time and effort next time.
- **Overly successful search:**
  - There may be too many similar assets, resulting in too much effort spent on analysis.
- **Fuzzy criteria:**
  - The criteria for what to search for need to be crisp enough so that an overly successful search is avoided, yet general enough so that not all viable candidates are ruled out.
- **Failure to search for nonsoftware assets:**
  - Failure to consider nonsoftware assets in your search, such as specifications, test suites, procedures, budgets, work plans, requirements, and design rationale, will reduce the effectiveness of any mining operation.
- **Inappropriate assets:**
  - Assets recovered from a search may appear to be usable but later turn out to be of inferior quality or unable to accommodate the scope of variation required.
- **Bad rehabilitation estimates:**
  - Initial estimates of the cost of rehabilitation may be inadequate, leading to escalating and unpredictable costs.

Mining Assets Practice Risks
(Corporate Risks)

- **Lack of corporate memory:**
  - Corporate memory may not be able to provide sufficient data to utilize the software asset effectively.
- **Inappropriate methods:**
  - The wrong reengineering methods and tools may be selected, leading to schedule and cost overruns.
- **Lack of tools:**
  - Tools required for the mining effort may not be integrated to the extent necessary, leading to risky and expensive workarounds.
- **Turf conflicts:**
  - Potential turf conflicts may undermine the decision process in selecting between similar candidate assets. Or, a repository of assets may be off limits for political or organizational reasons.
- **Inability to tap needed resources:**
  - There may be an inability to free resources from the group that originally created the component to rehabilitate or renovate it.

Software System Integration

Software Engineering Practice Area

- Software system integration refers to the practice of combining individual software components into an integrated whole
- Software is integrated when components are combined into subsystems or when subsystems are combined into products
- An incremental approach to integration decreases risk
  - Problems encountered during software integration are often the most complex, incremental steps reduce this risk
- Significance of "Interfaces"
  - Simplistic programming language definition too incomplete
    - This definition of "interface" may let two components compile together successfully, but only the Parnas definition (which subsumes the simpler one) will let two components work together correctly

Product Line aspects of Integration

- 2 points of integration
  - In a product line effort, software system integration occurs during the installation of core assets into the core asset base and also during the building of an individual product
- Range of effort for integration
  - Considerable coding may be involved to bring together the right core components into a cohesive whole
  - Generate final products by supplying the actual parameters specific to the individual product requirements and then launching the construction tool
  - In practice, organizations fall in the middle of this range

Specific Practices for Integration

- **Interface languages:**
  - Programming languages such as IDL allow you to define machine-independent syntactic interfaces
- **Wrapping:**
  - Wrapping, described as a specific practice in the "Mining Existing Assets" practice area, involves writing a small piece of software to mediate between the interface that a component user expects and the interface that the used component comes with.
- **Middleware:**
  - An especially integrable kind of architecture employs a specific class of software products to be the intermediaries between user interfaces on the one hand and the data generators and repositories on the other
- **System generation**
  - all (or most) of the product line variability is known in advance
- **FAST generators**
  - The Family-Oriented Abstraction, Specification, and Translation (FAST) process begins by explicitly identifying specific commonalities and variabilities among potential family members and then designing a small special-purpose language to express both. The language is used as the basis for building a generator

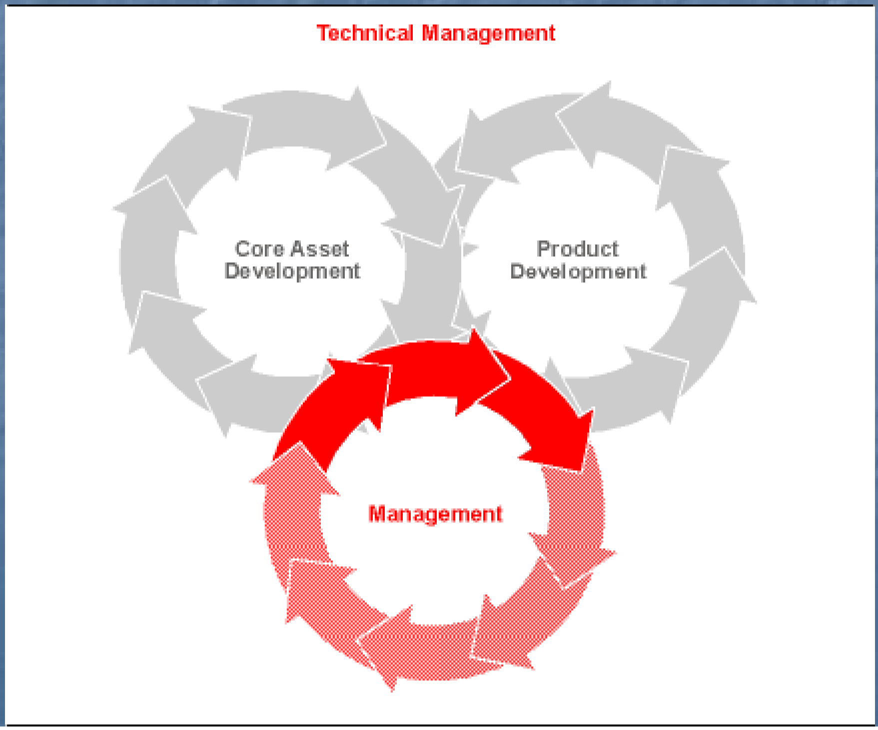
Software System Integration
Practice Risks

- **Natural-language interface documentation:** Relying too heavily on natural language for system interface documentation and not relying heavily enough on the automated checking of system interfaces will lead to integration errors. Natural language interfaces are imprecise, incomplete, and error-prone. Carrying forward in the face of undetected interface errors increases the cost of correcting such errors and increases the overall cost of integration. Automated tools, however, are more oriented to syntactic checking and are less effective at checking race conditions, semantic mismatch, fidelity mismatch, and so on. Some interface specifications must still be done largely with natural language and are still error-prone.
- **Component granularity:** There is a risk in trying to integrate components that are too small. The cost of integration is directly proportional to the number and size of the interfaces. If the components are small, the number of interfaces increases proportionally, if not geometrically, depending on the connections they have to each other. This leads to greatly increased testing time. One of the lessons of the CelsiusTech case study was that "CelsiusTech found it economically infeasible to integrate large systems at the Ada-unit level" [Brownsword 96]. Although the component granularity is dictated by the architecture, we capture the risk here, because this is where the consequence will make itself known.
- **Variation support:** There is a risk in trying to make variations and adaptations that are too large or too different from existing components. When new components or subsystems are added, they must be integrated. Variations and adaptations within components are relatively inexpensive as far as system integration is concerned, but new components may cause architectural changes that structure the product in ways that cause integration problems.

End of Lecture 2

Technical Management Practice Areas

- Management of the development and evolution of both core assets and products
- Technical management practices are carried out in the technical activities represented by the core asset and product development



Technical Management Practice Areas

- In alphabetical order, the practice areas in technical management are as follows:
  - Configuration Management
  - Data Collection, Metrics, and Tracking
  - Make/Buy/Mine/Commission Analysis
  - Process Definition
  - Scoping
  - Technical Planning
  - Technical Risk Management
  - Tool Support

  We will discuss Several of these Practice Areas

Configuration Management

Technical Management Practice Areas

- *The purpose of Software Configuration Management is to establish and maintain the integrity of the products of the software project throughout the project's software life cycle.*
  - *Software Configuration Management involves identifying configuration items for the software project*
  - *Controlling these configuration items and changes to them*
  - *Recording and reporting status and change activity for these configuration items [SEI]*

Configuration Management

Technical Management Practice Areas

- Successful CM requires a well-defined and institutionalized set of policies and standards that clearly define:
  - the set of artifacts (configuration items) under the jurisdiction of CM
  - how artifacts are named
  - how artifacts enter and leave the controlled set
  - how an artifact under CM is allowed to change
  - how different versions of an artifact under CM are made available and under what conditions each is allowed to be used
  - how CM tools are used to enable and enforce CM
- These policies and standards are documented in a CM plan that informs everyone in the organization just how CM is carried out

Unique Aspects of CM for Product Lines

- CM for product lines is more complex than it is for single systems
  - In single-system CM, each version of the system has a configuration associated with it that defines the versions of the configuration items that went into its production
  - In product line CM, there must be a configuration maintained for each version *of each product*
  - In single-system CM, each product with all of its versions may be managed separately.
  - In product line CM, this is untenable because the core assets are used across all products. Hence, the entire product line is usually managed with a single, unified CM process.
- Product line CM must control the configuration of the core asset base and its use by all product developers
  - It must account for the fact that core assets are usually produced by one team and used in parallel by several others
  - Single-system CM has no such burden: the component developers and the product developers are the same
- Only the most capable CM tools can be used in a product line effort
  - Many tools that are adequate for single-system CM are simply not sufficiently robust to handle the demands of product line CM

Product Line Demands Upon CM

- **Parallel development:**
  - In a product line development, there are occasions when the same items are being worked on by different people/groups and for different purposes. This imposes a dual requirement on the CM process—to allow separate strands of work to continue without conflict and to provide for their ultimate consolidation. Thus, when shopping for a CM tool, look for branch and join capabilities to manage and track the version history of an artifact.
- **Distributed engineering:**
  - Organizations that develop product lines might have more than one development/maintenance site. This requires CM to support distribution, possibly via a network. Depending on the speed of the network connection, this may imply replication of configuration items, which would require the CM to keep replicated configuration items consistent over the network. In a distributed development/maintenance environment, it is also likely that the different groups will be working in different development environments. In that case, CM will need to support heterogeneous environments, which will require import/export features.
- **Build and release management:**
  - Build management enables developers to create a version of a product, which can be anything from a single component to a complete customer solution for the purpose of testing and/or integration. Release management builds the final customer solution, which also includes instantiation of the developing and testing environment. In a product line context, release management includes the release of core assets to product developers. When shopping for CM tools, make sure the one you buy can help you build releases.
- **Change management:**
  - An authorized group must analyze carefully any changes proposed to artifacts that are under configuration control. The resolution of the proposals must be communicated, and any resultant changes must be planned, assigned, tracked, and broadcast. Changes in core assets need to be weighed carefully for their impact on the entire product line. Requests for changes in core assets can come from any product team or from the core asset team.
- **Configuration and workspace management:**
  - The individual handling of artifacts is also important to manage. Configuration and workspace management specifies what a configuration is; this includes the testing and support environment (for instance, a compiler and a debugger) and how users can create their own workspaces or views when working on a configuration. Whether or not your CM tool helps you with this, remember to carry along the environmental information with each artifact.

Product Line Demands Upon CM

- **Process management:**
  - Defined processes are essential for a mature CM capability. Especially in a product line environment, the CM must define the process life cycles for the configuration items and set up appropriate change control and authorization policies for product element modifications. CM processes need to be reviewed and improved as the product line effort matures. Any changes in the CM process need to be managed and rolled out carefully to the entire product line organization. The defined process must address:
  - **life-cycle management:** The life cycle is defined for every type of configuration item, assigning states and possible transitions to it. Changes in any item need to be analyzed, authorized, planned, implemented, documented, reviewed, tracked, and communicated. Also included are: a list of actions (such as notifying a set of interested parties) that need to occur when an item transitions from one state to another; change authorization policies that define how changes are authorized for an item; and closure rules that specify when an item (such as a change request) is closed.
  - **roles and responsibilities:** There are roles and responsibilities associated with each configuration item. For example, roles include owner (who has the responsibility for the artifact), reviewer (who analyzes changes in the artifact), implementer (who makes the changes), and so on. CM must support the definition and management of product line roles, many of which are nontraditional.
  - **configuration item identification and attributes:** Specific configuration items need specific names and information (labels). The information needed for a core component (for example, the products in which the core asset is currently used) differs from that needed for product-specific components. There might also be a requirement within a particular product line development to assign special attributes (for instance, has a proxy). The product line CM system has to have a means of applying this kind of attribute customization.
  - **repository management:** This facilitates the storage of the configuration items with version management and branching, and their attached information, as well as providing a comprehensive query capability for accessing all the information in the repository.
- Product line CM must also support the process of merging results either because new versions of core assets are included in a product or because product-specific results are introduced into the core asset base. Finally, since introducing changes may affect multiple versions of multiple products, you'll want your CM system for a product line to deliver sound data for an impact analysis to help you understand what impact a proposed change will have.

Specific CM Practices

- Identify the configuration items, components, and related work products that will be placed under configuration management.
- Establish and maintain a configuration management and change management system for controlling work products.
- Create or release baselines for internal use and for delivery to the customer.
- Track change requests for the configuration items.
- Control changes in the content of configuration items.
- Establish and maintain records describing configuration items.
- Perform configuration audits to maintain the integrity of the configuration baselines.

CM Practice Risks

- **Process not sufficiently robust:**
  - CM for product lines is more complex than CM for single systems. If an organization does not define a robust enough CM process, CM will fail, and the product line approach to product building will become less efficient.
- **CM occurs too late:**
  - If the organization developing the product line does not have CM practices in place well before the first product is shipped, building new product versions or rebuilding shipped versions will be very time-consuming and expensive, negating one of the chief benefits of product lines.
- **Multiple core asset evolution paths:**
  - There is a risk that a core asset may evolve in different directions. This can happen by design to enable the usage of a core asset in different environments (operating systems, for example) or by accident when a core asset evolves within a specific product. The first case increases the complexity of the CM but might not be avoidable. Attention should be directed to the second case. If this cannot be avoided, the usefulness of the core asset base will be degraded.
- **Unenforced CM practices:**
  - Owing to the complexity of the total product line configuration, not enforcing a CM process can result in total chaos (much worse than for a single system).
- **Tool support not sufficiently robust:**
  - CM sophisticated enough to support a nontrivial product line requires tool support, and there is no shortage of available commercial CM systems. However, most of them do not directly support the required functionality to be useful in a product line context. Many of them can be "convinced" to provide the necessary functionality, but this convincing is a time-consuming task requiring specialized knowledge. If the organization fails to assign someone to customize the CM system to the needs within the product line development, the CM tool support is likely to be ineffectual. Such a person needs to have both a good understanding of the product line processes and a solid grounding in CM

Data Collection, Metrics, and Tracking

Technical Management Practice Areas

- **The manager of an effort sets goals, defines objectives that satisfy those goals, and then creates a plan and applies resources to achieve those objectives**
- **Initiation Phase**
  The initiation phase is a planning activity that involves the following steps:
  - Designate the goals that will be tracked.
  - Define the metrics that will be used to track the progress toward those goals.
  - Identify the data that must be collected in order to derive those metrics.
  - Characterize the expected results and issues that may be discovered, based on any foreseen risks.
  - Specify how the data will be collected, when the data will be collected, and by whom.
- **Performance Phase**
  The performance phase carries out the plan, and involves the following steps:
  - Collect the specified data.
  - Analyze and translate the collected data into metrics and compare them against the expectations that were characterized during the initiation phase.
  - Determine the actions that are needed to remedy any discovered issues.
  - Confirm whether those actions were appropriate for addressing those issues.

Product Line Unique Measurement

- In a product line data collection needs to provide information from three perspectives
  - Not just the single perspective of product development.
  - Recall the three essential activities of product line development:
    - Core Asset Development
    - Product Development
    - Management
- Core asset development
  - comprising efforts to produce reusable assets and the supporting infrastructure for their use
- Product development
  - comprising efforts to produce individual products for customers
- Management of the overall product line
  - Including the strategic planning and direction of a total product line enterprise
- A product line manager is concerned with tracking whether the overall multiproduct effort is efficient and effective and is progressing properly toward achieving its strategic goals and satisfying the product line's production constraints

Data Collection and Metrics Specific Practices

Product Line Indicators and Measures [Zubrow 03]			
Goal	**Product Line Manager**	**Asset Development Manager**	**Product Development Manager**
Improved Performance	• total product development cost • productivity • schedule deviation • time-to-market • trends in defect density • number of products (past, current, future) • time spent on life-cycle activities	• cost to produce core assets • cost to produce infrastructure • schedule deviation • defect density in core assets • number and type of artifacts in asset library	• direct product cost • defect density in application artifacts • percent reuse
Compliance	• mission focus • process compliance	• mission focus • process compliance	• process compliance
Increased Effectiveness	• return on investment • market satisfaction	• core assets utility • core assets cost of use • percent reuse	• customer satisfaction

Data Collection and Metrics Practice Risks

- **Metric mismatch:**
  - Metrics that are not based on product line, subordinate core asset, or product development goals will result in wasted effort spent collecting data that do not contribute to management decision making.
- **Goals without metrics:**
  - Goals that have no associated metrics will result in managers being unable to detect any issues that hinder the achievement of those goals until an unacceptable expenditure of work or time has been incurred.
- **Measurement not aligned:**
  - Any measurement activity that is not integrated into the product line process will result in data collection that does not mesh properly into other product line activities and that leads to inaccurate results that either hide legitimate issues or raise false issues.
- **Costly metrics:**
  - Metrics that are too costly or difficult to obtain will result in failure to track progress, which in turn will result in failure or delays in detecting problem issues or interference with the effort's primary work.

Process Definition Practices

Technical Management Practice Areas

- Whether or not a formal model is built from a defined process, its definition should be represented in a form that is understandable by humans and should be clear and complete enough to satisfy the following goals:

- Facilitate human understanding and communication
  - Enable communication about and agreement on the software process.
  - Provide sufficient information to allow an individual or team to perform the intended process.
  - Form a basis for training individuals to follow the intended process.

- Support process management
  - Develop a project-specific software process to accommodate the attributes of a particular project, such as its product or organizational environment.
  - Reason about the attributes of software creation or evolution.
  - Support the development of plans for the project.
  - Monitor, manage, and coordinate the process.
  - Provide a basis for process measurement, such as the definition of measurement points within the context of a specific process.

- Support process improvement
  - Reuse well-defined and effective software processes on future projects.
  - Compare alternative software processes.
  - Estimate the impacts of potential changes in a software process before putting them into actual practice.
  - Assist in the selection and incorporation of technology (tools, for example) into a process.

Product Line Unique Process Definition Aspects

- Product line software engineering requires EXTREME cooperation
  - Because of the plurality of products and of groups cooperating to develop those products, the entire apparatus will work only if everyone does his or her job within agreed-upon parameters.
    - For example, no one is allowed to change core assets unilaterally
- Besides configuration management, two other prime examples of product line processes are:
  - The operational concept for the product line, such as the one embodied in a product line concept of operations.
    - The concept of operations is essentially the expression of a process, and process definition skills must be brought to bear to build an operational definition that everyone can follow, that can be improved, and that will serve the goals of the product line.
  - The attached processes that help product builders instantiate core assets for specific products
    - These processes must account for a wide variety of core assets and variabilities and accommodate the different role-players who are going to carry them out.

Product Line Specific Process Definition Practices

- Electronic process documents for the user of a process
  - A Web-based process handbook or an electronic process guide.
  - These documents allow process definitions to be focused or presented in different ways to shield the process user from scores of unnecessary details.
  - Instead, the electronic documentation displays a narrowed view that describes at any specific time the process steps that should be performed
  - In general, role-specific views of a process decrease the complexity of process definition and allow one to focus on those aspects of the process that are relevant to a specific user
- Integrated process support environments utilize computer systems to automate some of the required process steps
  - E.g. informing a quality assurance organization that a document is ready for review or sending a change request to core asset developers
- A generic process definition will obviate the need to define a comprehensive product line process that would take into account all the possible variabilities—a daunting task
  - A generic process represents a class of processes for building a class of products. In other words, a product line process represents a family of processes
  - The variations in the processes within a family reflect the variations in the products

Process Definition Practice Risks

- **Process mismatch:**
  - There is a possibility of a process mismatch among a number of factors, such as the organizational structure, the organizational culture, the process that is employed, employees' experience and expertise, and the market to which the process is applicable. For example, the defined process may be too complex for the organization, resulting in detailed practices that are not followed. On the other hand, the process may be too simplistic and thus too general and at too high a level to provide practical guidance.
- **Process doesn't address product line needs:**
  - The process may not accommodate a bidirectional flow between core asset development and product development processes. This flow is necessary for product line success.
- **Inadequate process support:**
  - Processes have to be supported within an organization, especially newly introduced ones. Lack of support (such as training, motivation, templates, and examples) will lead to rejection of the process.
- **Uneven process quality:**
  - Because of divergent goals, skills, and backgrounds, there may be uneven quality in the contributions of the core asset teams and product development teams. This may also result in a lack of harmonization in the processes for the different teams.
- **Lack of buy-in:**
  - The organization may not buy into process definition, resulting in failure to adopt it.
- **Dictatorial introduction:**
  - One organizational unit mandating the processes that another must follow is likely to result in resentment and failure to adopt them.

Scoping

Technical Management Practice Areas

- Scoping is an activity that bounds a system or set of systems by defining those behaviors or aspects that are "in" and those behaviors or aspects that are "out."
  - All system development involves scoping; there is no system for which everything is "in”
- Scoping is a fundamental activity that will determine the long-term viability of the product line
- The scope definition identifies those entities with which products in the product line will interact
  - It also establishes the commonality and bounds the variability of the product line
- The goal of the scope definition is to draw the boundary between in and out in such a way that the product line is profitable

Scoping Drivers

- The prevailing or predicted market drivers, obtained through "Market Analysis" practices
- The nature of competing efforts, obtained through "Market Analysis" or "Understanding Relevant Domains" practices
- Business goals that led to embarking on a product line approach, obtained through the "Building a Business Case" practice area
  - An example of a scope-setting goal is the merging of a set of similar, but currently independent, product development projects to save cost
- Scopingis a way to help inform decision-making
  - Given a product opportunity, should the organization bring that product into the product line family?
  - However mature product line organizations know how to use their scope to make their own product opportunities

Product Line Scoping Practices

- **Examining existing products:** Conducting a thorough study of existing products helps identify commonality across a potential product line and identifies the types of differences that are likely to occur.
  - Identify existing products similar to those that will be part of the product line.
  - Gather any available documentation and conduct product demonstrations.
  - Conduct oral or written surveys of the current developers, users, and maintainers of these products and product experts.
  - Identify the products' capabilities, structure, and evolution, and any other relevant factors about them.
  - Determine which elements of these products should be considered part of the product line.
- **Conducting a workshop to understand product line goals and products**
  - the business goals to be satisfied by the product line
  - the mapping of product line business goals to the organization's business goals and to users' needs
  - descriptions of current and potential future products that will constitute the product line
  - essential product line core assets that may include platforms, standards, protocols, and processes

Scoping practice Risks

- **Scope too big or too small:**
  - For a product line to be successful, its scope must be defined carefully. If an attempt is made to encompass product members that vary too widely, the core assets will be strained beyond their ability to accommodate the variability; economies of production will be lost; and the product line will collapse into the old-style, one-at-a-time product development effort.
- **Scope includes the wrong products:**
  - Commonalities and variations across current and future systems will include functional requirements, user concerns, and interactions with external systems. In addition, these commonalities and variations will include system qualities, performance issues, and technology evolution.
- **Essential stakeholders don't participate:**
  - The specific practices in scoping require participation from a wide range of stakeholders. These stakeholders will include management, developers, customers, users, methodologists, and subject-matter experts.

Other Technical Management Practice Areas

- Technical Planning
- Risk Management
- Tool Support

Organizational Management

Practice Areas

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

We will discuss a few of these key practice areas

Building A Product Line Business Case

Organizational Management Practice Areas

- **A business case addresses the key questions that an organization faces when planning major changes in its current ways of doing business:**
  - **What are the specific changes that must occur?**
  - **What are the benefits of making the change?**
  - **What are the costs and risks?**
  - **How do we measure success?**
- **An effective business case must convince management**
  - **That the investment is financially sound**
  - **The PLA is realistic for the organization**
  - **The PLA is aligned with other business strategies**
  - **THere is a clear course of action for putting the change into effect**
- **Business case results are often summarized using several well-defined financial metrics such as net cash flow, discounted cash flow, internal rate of return, and payback period**

Business Case Description -2

- The business case documents how closely aligned the opportunity is with established business goals for such things as:
  - reduced time-to-market
  - reduced cost
  - higher productivity
  - improved quality
  - increased customer base or bigger market share
  - ease of upgrades
- There is no standard organization for a business case, but it should address the following
  - Deciding what to do: list any assumptions (market conditions, organizational goals, and so on), develop alternative approaches, and then either choose one or decide to build a comparison.
  - Estimating the likely costs and potential risks of all alternatives.
  - Estimating the likely benefits contrasted with the current business practice.
  - Developing a proposal for proceeding.
  - Closing the deal: how to make final adjustments and proceed to execution.

Product Line Specifics for Business Cases

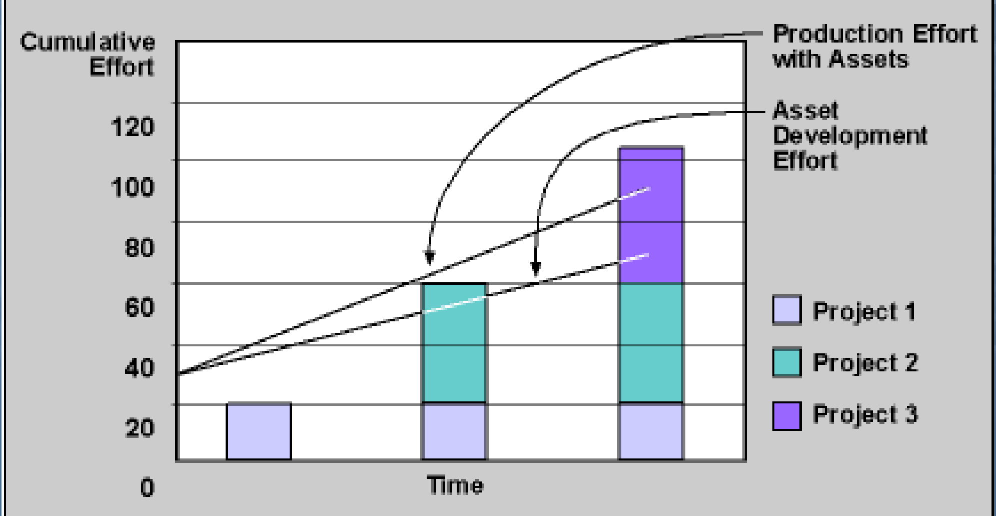
- A business case in a product line context can serve one of two purposes
  - The first is to justify the effort to adopt the product line approach for building systems
  - The second is to decide whether or not to include a particular product as a member of a product line.
- The business case should be
  - Agreed upon
  - Documented
  - Communicated to the entire organization
  - Validated by market analysis and organizational experience and expertise
- Here, the initial go/no-go decision answers the question:
  - "Do we build the set of products we're considering as a product line or not?"
- As part of the business case analysis, the organization determines
  - How many products are likely to be built in the product line over a certain time
  - Who the customers will be
  - Whether a product line approach compares favorably with other business opportunities

Product Line Specifics for Business Cases

- Key Questions
  - Do we have the right capability and resources to launch a product line?
  - Can we leverage our domain understanding to provide a unique opportunity and create market demand for our product line?
  - What are the financial and business consequences of adopting a product line approach?
- A Mature Product Line based Org Morphs the questions
  - Do we have the right capability and resources to <u>build this product as a member of our product line</u>?
  - Can we leverage our domain understanding to provide a unique opportunity and create market demand <u>for this product</u>?
  - What are the financial and business consequences of <u>including this product in our product line</u>?

Impact of Business Case On Core Asset Development

- The Business Case IS a Core Asset and should be managed
- There is an initial start-up cost
  - Shown in the figure as 30 units of effort, for moving to a product line approach.
  - In addition to costs for developing core assets, the business case must include the cost of adopting processes for product lines
    - including the costs of training, incentives, and tool development or procurement. In the figure, this cost is shown as accruing even before the launch of the first project.
- With each successive project, core assets must be maintained and enhanced, and new core assets added. Thus, the cumulative cost for developing core assets increases over time
- In the figure, the "Production Effort with Assets" line represents the cumulative effort of developing all three projects shown
  - Project cost includes start-up cost, the cost of enhancing the core asset base for that project, and the cost of project-specific development.



Based on: Weiss, et.al. Software Product Line Engineering

Product Line Business Case Practices

- **Estimating the likely costs and benefits:**
  - For each alternative, the organization makes reasonable cost estimates. These costs may be accrued at different times:
- **Initial costs:**
  - When the product line's core assets are developed and the initial products are fielded.
- **Incremental costs:**
  - Whenever the product line is extended with new core assets. The extensions include improvements within the existing scope or an extension of the scope itself.
- **Product development costs:**
  - Costs associated with using core assets in developing products.
- **Annual costs:**
  - Upgrades and annual maintenance costs to fix defects.
- The Structured Intuitive Model for Product Line Economics (SIMPLE) is an economic models specifically geared for software product lines

Product Line Business Case Practice Risks

- **Insufficient data:**
  - It is usually necessary to set cost expectations early and then refine the cost information as the project progresses.
- **Unreliable historical data:**
  - Most cost development methods rely on good historical data, either from within the organization or from industry.
- **Approaches that fail to work across organizational boundaries**
- **Uncertain market conditions:**
  - What will be the cost of transition? Who will use the core assets? How many products will be needed per year? How long will the product line last?
- **Management indecision:**
  - The group developing the business case must understand the audience. This audience must include those who can make the final go/no-go decision for proceeding on the proposals contained in the business case
- **Shift in organizational goals and needs:**
  - If the goals and needs of the organization have shifted during the preparation of the business case, the results may not be useful or meaningful.

Customer Interface Management

Organizational Management Practice Areas

- Managing the customer interface will require your organization to:
  - Identify the groups or individuals who are responsible for interfacing with customers
    - What defines the customer interface and whom does it involve?
    - Who is the customer?
  - Define clearly the roles and responsibilities of the designated customer representatives
  - Ensure that customer representatives are trained properly in their roles and responsibilities
  - Implement effective processes to govern the organization's customer interactions and ensure clean interfaces among customer representatives

Customer Interface Management

- The major elements of the customer interface that must be managed are as follows:
- What the customer will see
  - Who are the customer representatives and what are their customer interface responsibilities?
  - What are the standard product offerings and the preplanned feature variability?
  - What are the corresponding cost, schedule, and quality benefits?
  - What is the product line strategy for future features and evolution?
- What the ground rules are for transacting business
  - What protocol must be followed and what policies and procedures apply?
  - How are customer requirements to be negotiated and managed?
  - How will a disciplined interface with the customer be enforced?

Customer Interface

- *Marketer: "You know, if you were to relax this requirement over here, and drop that little one over there, and change this one over here just a bit, then we could build you a system that's in our software product line."*
- *Customer: "I see. And this matters to me because . . .?"*
- *Marketer: "Because if we build your system from scratch, it will cost you $4 million, take two years to deliver, and your software will be unique. If you take a member of our product line, it will cost you $2 million, be ready in six months, and your software will be the same that is currently running reliably for 16 other customers. But of course, it's entirely up to you."*

- The interface with customers becomes market driven and no longer focuses on an individual customer's specialized requirements
  - Specialized requirements can be accommodated, but must be considered individually and have special cost and schedule implications to which both parties must agree.
- Strict product line organizational interfaces are enforced
- Customers choose from standard product offerings
  - Relinquishing some flexibility in return for cost and time-to-market advantages
- Customers may form user groups to give the *market* a voice and drive requirements for product evolution jointly
- The organization can consistently deliver products of predictable quality at predictable cost and delivery time

Customer Interface Specific Practices

- Communicate the product line strategy to the customer
- Establish an overarching customer interface process
  - For developing work proposals and negotiating new contracts
- Train product line marketers and product managers
- Provide centralized product support to customers
- Establish a user's group, or other liaison means
  - To assist the product line organization in identifying and prioritizing its customers' emerging and long-term future needs

Customer Interface Practice Risks

- **Failure to recognize the extent of the customer interface and its effects**
- **Organizational discontent and resistance in transitioning to a new business paradigm**
- **Marketers promise the world and fail to point out any tradeoffs that are involved**
- **Marketers and product managers are insensitive to specialized customer requirements**
- **Customers fail to recognize benefits properly and see only the loss of flexibility**
- **The product line organization releases a scheduled product upgrade to all customers that includes unannounced changes that cause sporadic problems for customers**
- **Failure to enforce the interface**
- **Improperly trained customer interface staff**
- **Customers with their own agendas dominate user group forums so that other users or customer communities are not heard**

Product Line Architecture Funding

Organizational Management Practice Areas

- The key funding question for a product line organization is how to fund the core assets which will be used across several products, most of which will probably not come into existence until long after the core assets are initially put in place
- A software product line requires an investment to get it off the ground
- This funding must be sufficient so that the core assets can be of high quality and have the appropriate applicability
- The "up-front" work on a product line often has to be accomplished in parallel with ongoing operations
  - As a result, new or innovative sources of funding for the organization are often required for the product line launch

Funding Specific Practices

General Applicability of Funding Strategies to Product Line Activities							
		Activities to be Funded					
		Planning and Analysis	Product Line Development		Product Line Sustainment and Evolution	Product Development Sustainment and Evolution
Funding Strategies			Infra-structure Development	Asset Development		
1	Product-specific funding (individual customer, for example)			XX		XXX
2	Direct funding from corporate sponsor/program	XXX	XX	XX		
3	Product line organization's discretionary funds	X	XXX	X	X	
4	First product (project) funds effort	XX	XX	XX	X	XXX
5	Multiple projects banded together to share costs	XXX	XX	XXX	XX	X
6	Taxing of participating projects		X	X	XXX	
7	Product-side tax on customers			X	XXX	
8	Fee based on core asset usage				XXX	
9	Prorated cost recovery		X	XX	X	X

Funding Practice Risks

- **Inflexibility of the organization's fiscal infrastructure**
- **Instability of initial management commitment**
- **Waning management commitment**
- **Externally imposed fiscal constraints**
- **Lack of strategic focus**
- **Inadequate funding**

Structuring the Organization for Product Lines
Organizational Management Practice Areas

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

Organizational Structure Needs

- An organizational structure should be chosen to at least determine which unit or units:
  - 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.
  - 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
- A first-order choice that must be made when choosing an organizational structure is where to house the people who develop, maintain, and evolve the product line's core assets. Typically, organizations take one of two approaches: either they form a separate unit for the developers and maintainers of core assets, or they house that effort in the same unit or units that build products.

Deciding Structure

- **The size of effort and 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, a dedicated core asset group will help.
- **New development or mostly legacy-based development:**
  - In product line efforts wherein the core assets are built largely from legacy components, it makes more sense to have product developers (who will probably be more familiar with the legacy assets) be responsible for making the legacy assets more generic to fit the scope of the product line.
- **The funding model:**
  - Funding a component engineering 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 component group when the product development groups are adding product-specific features to the core assets.
- **The high or low effort for 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 component engineering 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.
- **Parallel 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.

Structure Practices and Risks

- We will discuss a specific example in the next lecture
- Risks:
  - **The current level of organizational stress**
  - **The implementation history**
  - **Sponsorship**
  - **Resistance management**
  - **Culture**
  - **Change agent skills**

End of Lecture 3

End of Material that will be basis of homework