Technical Report
CMU/SEI-96-TR-010
ESC-TR-96-010

Carnegie-Mellon University
Software Engineering Institute

Investment Analysis of Software Assets for Product Lines

James Withey

November 1996

Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders.

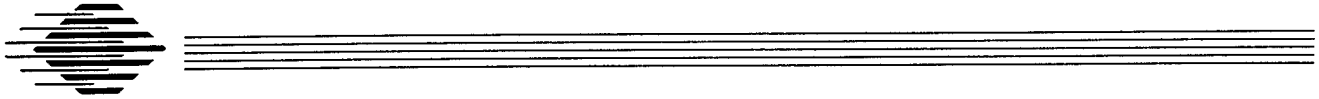
In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state, or local laws or executive orders. However, in the judgment of the Carnegie Mellon Human Relations Commission, the Department of Defense policy of, "Don't ask, don't tell, don't pursue," excludes openly gay, lesbian and bisexual students from receiving ROTC scholarships or serving in the military. Nevertheless, all ROTC classes at Carnegie Mellon University are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2056.

Obtain general information about Carnegie Mellon University by calling (412) 268-2000.

Technical Report
CMU/SEI-96-TR-010
ESC-TR-96-010
November 1996

Investment Analysis of Software Assets for Product Lines



James Withey

Product Lines Systems Program

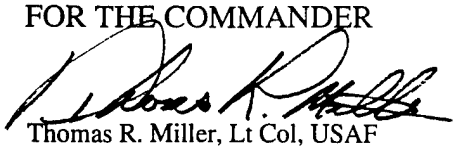
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER



Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Pittsburgh, PA 15212.
Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145. Phone: (703) 274-7633.

Table of Contents

List of Figures

Preface

Purpose

Group, product line, and program managers are faced with allocating resources to projects. Should all resources be dedicated to meet near-term deliverables? Or should some be siphoned off to build software assets that may improve quality, flexibility, and reduce cost and time-to-market of future products in the product line? These managers also have to determine which assets to buy or build. The choices are many, ranging from reusable code components to design models to application generators, and each has a different risk and cash flow profile.

This report introduces an approach that will help managers make these allocation decisions. The report outlines a planning and communication tool for analyzing investments in software assets for product lines.

The goal of the approach is to provide managers a means to determine whether to build software assets for a product line. It will help managers use marketing forecasts to recognize the software assets proposed by engineering that have the highest economic leverage across the variety of products.

Although the report is not a guidebook, the concepts, criteria, and investment modeling techniques will be useful in making and justifying proposals for funding. The concepts are drawn from the fields of microeconomics, corporate finance, marketing, R&D technology management, and software reuse.

Details about the approach will evolve through data collection. Next steps include working with organizations that have developed assets for product lines. We want to understand the risks and factors that influence asset selection and economic outcomes. The goal is to have a better understanding of the relationships between an organization's business environment, the software assets it develops, and economies of scope.

To give the reader a sense of how the concepts are applied in practice, a hypothetical product line and assets are introduced; the reader follows the analysis of two assets proposed for a pedestal system product line.

Acknowledgments

Many people contributed to this report. Mark Bell provided systems expertise and effort estimates for the pedestal system example. He helped me model the system behavior and identify common system constraints for a variety of products. Professor Fallow Sowell at the CMU Graduate School of Industrial Administration gave advice on econ-

omies of scope during the early stages of writing this report. Professor Vassant Naik at the University of British Columbia reviewed an earlier draft, and gave invaluable advice on the investment model.

I wish to thank Paul Clements who actively listened to impromptu chalk board talks at a moment's notice while the concepts in this report were being developed. I also want to thank Jorge Diaz-Herrera for exciting discussions and for sharing with me a taxonomy of software design techniques. Many thanks to Sheila Rosenthal for the extensive literature searches and article retrievals.

John Bergey, David Bristow, Lisa Brownsword, Sholom Cohen, David Dikel, Mike Mattison, Ken McNulty, and Linda Northrop reviewed earlier drafts, and made many suggestions that sharpened the report. Kimberly Brune in Technical Communication served as technical editor. Thanks to you all.

This report benefits greatly from the pioneering work of engineers and economists studying flexible manufacturing systems (FMS). FMS encompass group technology[1] and programmable assembly robots. In combination, the technologies free manufacturers from having to produce large quantities of the same part before showing a profit; manufacturers are in some cases able to produce parts profitably in quantities of one — a case analogous to software. In studying the conditions and factors favorable for capital investment in FMS, economists and engineers developed strategies, concepts, and decision variables that I found applicable to the software engineering domain.

1. Group technology: short production lines (cells) that produce similar but different parts that can be used in a large variety of products.

Chapter 1 Introduction

1.1 Need

Managers can no longer afford to perpetuate the development and maintenance of software using a labor-intensive, craftsmanlike process. As software becomes more prevalent in products, the costs and throughputs of this process affect an organization's ability to satisfy customers and compete globally. If the goal is to improve an organization's responsiveness to evolving customer needs while reducing costs and/or increasing profits, then managers must transform software development and maintenance into an engineering process in which pre-developed, intermediate, and often incomplete solutions — assets — are used to build and modify software in a product line. Rather than treating each product as an isolated development project, managers must invest strategically in software assets to gain competitive advantage in the battlefield or the marketplace.

This report outlines an approach for analyzing the business value of software assets used in a product line. It introduces concepts and techniques to help sort out the issues involved in making an investment decision.

Suppose you are a manager responsible for four related products. You estimate that the software portion of the first product will cost $600 thousand to develop. The software in the next three products will be modifications of first product and will cost between $250 and $300 thousand each. Then a senior engineer shows you the following data based on early reuse experiences at IBM, NEC, Toshiba, GTE, AT&T, and HP [HP 93]:

- 1.5 to 2 times improvement in time-to-market
- 2 to 5 times reduction in maintenance costs
- 5 to 10 times improvement in quality
- 12 to 15% reduction in development costs at reuse levels of 50 to 80%

She explains that these results were accomplished by defining reusable components and devising processes and tools for composing new applications using these components. She estimates that the cost of building reusable software may total $950 thousand for the first product, and insists on technical grounds that the total cost on products two through four will be reduced by more than 25%. Although the first product will take longer to develop, she feels sure that the next three products will be delivered to the customer sooner.

You ask yourself, "Is it worth investing the extra time and money now to build software assets for reuse?" You answer, "Probably not." You then talk to marketing and learn that the product line is being expanded, and possibly 8 different products will be developed over the next three years. How does this information change your decision to build assets? Do you develop an architecture, components, and an engineering process for a line of products, or do you continue to craft each product independently, lifting code from the previous release?

An investment analysis that helps to recognize economically attractive reuse opportunities has not been developed. Because project managers perceive that the initial costs to develop a software asset are too high and it takes too long to break even, finding sponsorship to build software assets is difficult. Industry organizations that have developed assets had to create their own economic models to determine business value. SRI International reports that "economic and accounting issues are the biggest — and probably the least understood — of all barriers to successful component-based software development implementation" [Dewey 95].

Randall Macala from Boeing, reporting on the company's experiences developing software assets for a product line, recommends performing a rigorous business case analysis. The analysis

> "consists of a technical evaluation and an evaluation of the costs and schedule of the potential product line. The technical evaluation includes an analysis of the feasibility of engineering the domain and the evaluation of the amount of functionality shared by the product line's family members. The cost and schedule evaluation involves identifying what product line features will be needed and when, an estimated cost of development, and a predicted return on investment" [Macala 96].

He acknowledges that it is not easily done: "The business portion of the analysis remains incomplete" [Macala 96].

1.2 Investment Analysis

In general, four kinds of software estimating can occur in an organization:

- Senior managers estimate business performance (e.g., profit or market share) of a strategic mix of software-intensive products.
- Group, technology, and product line managers estimate the return on investment for improvements in software production. They choose investments considering the mix of products defined in the business strategy.

- Project managers estimate cost and schedule of a software project. Included in the estimate are the software assets that have been developed.
- Engineers estimate attributes (e.g. performance, complexity etc.) of a software product.

This report focuses on the investment decisions of managers who are responsible for the production of an existing or potential line of products. These managers usually have limited resources to allocate among competing investments in software process improvement, software technology, and software assets. Managers want to choose those projects with the greatest potential return on investment. Project proposals that provide an understandable and reliable analysis of return win over ones that do not.

Investment analysis is a process for defining and evaluating an investment. It involves specifying the investment, analyzing the uncertainties, constructing a spending strategy, and quantifying the costs and benefits. This analysis links the strategic and technical merits of an investment to its financial results.

Issues

Investment analysis of software assets for an existing or potential product line is not easy. Defining the investment (i.e., deciding which assets to make or buy) involves pooling insights from different groups. Criteria are needed to help a team screen candidate assets. Evaluating the investment involves estimating incremental costs and cost savings (or profit) over the lifecycle of the assets. If the costs and savings are uncertain, tasks should be added to the investment to develop the information. The investment should be structured so that a satisfactory resolution of the tension between management objectives and constraints, as shown in Figure 1-1, is achieved.

Objectives	Constraints
Shorter time-to-market	Limited funds
Lower costs	Limited time
Greater flexibility	Limited talent
Higher quality	Low risk

Figure 1-1: A Balance Between Objectives and Constraints

Identifying software assets with the highest economical potential is a technical challenge. Although between 45% and 85% of a new software product contains functionality that exists in prior products [Cusumano 91, Lanergan 84], not all this functionality should be developed as assets. The customization required for each product may be so much that little work would be eliminated or avoided using an asset. Nor should all as-

sets be software components. For example, when there is not a one-to-one mapping of functionality to a software module, a guidebook or a specification language may be more cost effective.

Because different assets have different properties, they will achieve the objectives in Figure 1-1 to different extents. Software assets vary in:

- the skills required to develop and use them
- the software lifecycle activities that they support
- the effort required to build and use them
- the range of product changes that they can accommodate

Their economic benefit depends on the organization's current skills and cost drivers, and on the variety of products planned in the product line. For example, a decision may be between purchasing middleware software (database and network service components) that is adapted using C++ and purchasing a rapid application development (RAD) tool. The RAD tool requires less skill to use and work is done at the application problem level. The middleware/C++ alternative is more flexible (i.e., it can be used in a greater variety of products), but it is also more complex and requires more knowledge of C++ and system protocols. Depending on current costs, availability of C++ skills, and the variety of products to build, one will be selected over the other.

Thus, to define the investment, criteria for screening the most promising candidates are needed. And since costs and benefits vary, a combination of different assets may be needed to achieve overall investment objectives.

There is also considerable technical and market risk in building and using software assets. For example, the solution common to members of the product line may turn out to be too immature or unstable. Or some assets will end up costing much more to develop and maintain than expected. If the market weakens, and some products are not developed, then some assets will not be used as much as originally expected, and this will affect the return on investment. Wayne Lim reports in one study that some work products developed at a Hewlett Packard facility produced a net loss. "The economic gain/loss ranged from a gain of 43.3 engineering days to a loss of 31.5 engineering days. The 31.5 engineering day loss was the result of fewer reuses and higher maintenance costs than expected" [Lim 92].

Given these uncertainties, an investment should be made incrementally. It should be divided into tasks that make costs commensurate with risks. Figure 1-2 shows a cumulative cost curve for an hypothetical investment. Typically the largest costs are not in the feasibility task where technical uncertainty is the highest. They are in the asset development task where risks are definable and therefore more manageable.

Each task tests an assumption about the investment and produces information. For example, one task could be collecting cost data on the current process and benchmarking other organizations; another task could be the design and evaluation of an architecture.

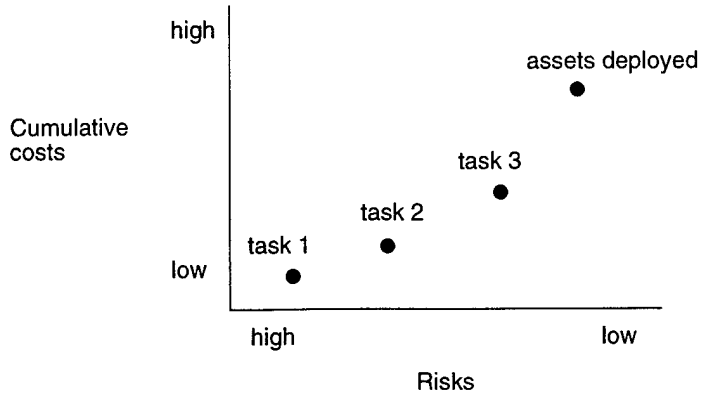


Figure 1-2: Cost/Risk Tradeoff

Given the information, a manager can choose to continue, delay, or stop investing more money. In other words, you do not move up the cost curve unless the results from a task reduce technical risk or show that market conditions are favorable. The plan of action consisting of these tasks is referred to as a *deployment strategy.* An optimum strategy is one that maximizes rightward movement and minimizes upward movement in Figure 1-2. An optimum strategy includes making management and organization changes as well as building software assets.

Most current reuse economic models that estimate return on investment adopt a static, accounting view of investing [Gaffney 92, Henderson-Sellers 93]. They assume that managers have only one decision: to start the investment. But in fact, they will execute different options as they become available throughout the investment. In traditional accounting, asset costs are amortized over the number of reuses or are treated as overhead on labor hours. Return-on-investment estimates are incorrect if the number of reuses and cost savings are at all uncertain. Since the analysis is performed after-the-fact, significant opportunities that would be identified while defining the investment are missed.

Most current models fail to establish a basis for comparing investments. To help managers allocate resources, investments are usually discounted by the opportunity cost of capital. This is the return that could be obtained if the cash used for the investment was placed instead in stocks and bonds with the same risks. By discounting all investments to their appropriate cost of capital, they can be compared to the same standard: the return in the equity markets.

An investment in software assets is a series of options that management exercise to receive future benefits or additional options. Completing one task successfully gives you the option to continue with the project and increases the chances of achieving the desired benefits.

Current approaches often fail to:

- provide criteria that help to compare candidate software assets
- consider other lifecycle assets besides code modules
- focus on cost drivers in the lifecycle. (Most focus on reducing the lines of code that have to be redeveloped from product to product, rather than reducing or eliminating the software development and maintenance activities that incur the most cost from product to product.)
- develop a strategy for managing technical risk
- incorporate future decisions that depend on market and technical outcomes
- establish a basis for comparison to other investments and account for variation in the opportunity cost of capital. (Depending on the risks, a different discount rate may be used.)
- consider the variety of products in the product line

Some economic models are designed to calculate the project costs to build or reuse software components [Boehm 95, Poulin 93], rather than compare software assets and estimate investment return. They answer questions such as, "What is the estimated cost to my project to build these components for reuse?" or "What is the savings from reusing these components?" They do not answer the question, "What is the return on investment from building and using these components in a product line?" Project costs are used in estimating return on investment, but the issues above still apply.

Approach

This report introduces an approach for defining and evaluating investments in software assets. To define the investment, a team answers the following questions:

- What software assets are likely to shorten time-to-market, increase quality and lower costs with minimum risk and expense? For example, would a code generator or a class of objects be a better investment for producing multiple versions of a user interface?
- What is the best way to phase in software assets so that exposure to uncertainties in asset technology and the market is reduced?

Evaluating the investment, a team answers the following question:

- Would the time, money, and engineering expertise used to acquire software assets for use in software development and maintenance be worth it? Are the benefits to the product line worth the expense and the risks?

The purpose of this approach is to enhance the abilities of managers to define asset development projects that minimize risk, to allocate limited resources to the right projects, and to reason about issues affecting product line objectives. The objective is to help managers make economically sound decisions regarding the development or acquisition of software assets for product lines.

The approach is a two-step process. First a portfolio of candidate assets is constructed based on a market and technical evaluation of products in a product line. Then a strategy for deploying the portfolio is constructed and modeled to estimate return on investment.

Portfolio analysis is used to optimize investment objectives and diversify risks. Portfolio analysis is used in marketing [McDonald 95], finance [Brealey 91], and R&D investment [Roussel 91]. Financial portfolio analysis involves evaluating the risk and return of different combinations of stocks to determine the combination that satisfies an investor's objectives. Marketing portfolio analysis places marketing segments in a matrix according to their attractiveness.

In this approach, we use portfolio analysis to guide the selection of software assets. Candidate assets are compared along dimensions that affect investment objectives, and a combination is selected that optimizes the potential for cost economies and shorter cycle time while minimizing risk. The analysis helps a team to reason about asset choices at an appropriate level of detail. For example, a manager may ask

- Why does it take so long to build this asset?
- If we changed the scope of the asset by making it less flexible, would the development budget be smaller?
- Why does this asset affect current costs less than another, when its scope is so much larger?

Dynamic discounted cash flow analysis is used to estimate the return of the portfolio. Instead of assuming a single cash flow scenario (a static analysis), this analysis involves laying out several decision paths and scenarios that may occur as the organization builds the portfolio and requisite organization infrastructure. The analysis involves defining the technical and economic uncertainties in the investment, and building a decision tree that includes the contingent decisions that management can make in the future and the probable cash flows for each decision. This decision tree models the deployment strategy developed by the team. The present value of the investment is determined by working backwards along branches of the tree from the future to the present, aggregating the cash flows for the best possible decision (given current information) at each joint in the tree. The cash flows are discounted by an appropriate risk-adjusted interest rate.

Context

This approach should be used by commercial organizations when the business goal is to introduce a variety of products more quickly than the competition, the market is established, and other organizations are competing to provide innovative features and services. In such situations, the source of profits (or stable budgets) is from software upgrades and new releases targeted to specific sets of customers. When new products and market growth are positively related (shaded region in Figure 1-3), building software assets for multi-use within and across product lines is a strategic investment.

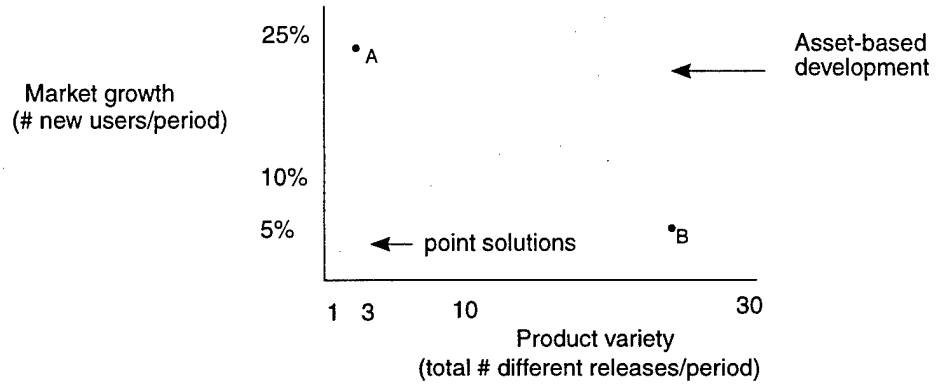


Figure 1-3: Conditions for Asset Development

Consider points outside the shaded region. At point A in Figure 1-3, the market is new: Growth is not driven by the introduction of a variety of products. Product sales are high for some other reason, most likely because the product provides a benefit or a service not yet offered by the competition: for example, an intelligent information browser for the Internet. The market is uncertain; marshalling the technology is more important than designing for variety. At point B, the market is saturated with choices. Additional products do not contribute greatly to growth or income. Unless development costs are very high, streamlining production costs through software assets will provide small or negative returns.

Government organizations should use this approach when a large portion of system functionality is replicated from product to product and flexibility in development is needed to adapt or extend software for use in new situations. This approach will help assess the value of investing in interoperability.

The approach described in this report only estimates the advantages that software assets add to production: lower costs, shorter cycle time, fewer defects, etc. The analysis is based on the extent that software assets are shared from product to product — the extent to which they produce economies of scope. Asset attributes that add value to the product, such as real-time fault tolerance or high security, are not included in the analysis.

1.3 Structure of Report

This report has five chapters.

Chapter 2 introduces terminology and concepts needed to frame the investment problem.

Chapter 3 walks the reader through a process for constructing a portfolio of software assets. A hypothetical product line is used to elaborate the concepts that are introduced.

Chapter 4 explains the approach to investment modeling and then estimates the return of an architecture and two components for a product line.

Chapter 5 summarizes the approach and raises some strategic issues.

Chapter 2 Background

2.1 Product Line Approach

Imagine a firm that sells products that provide geographical location information in digital form to users in diverse situations. The firm has a line of products for the taxi dispatch market, and is entering the car navigation market. The few car navigation products are loaded with features and are priced as high-tech novelty items, whereas the taxi dispatch product line has a lot of depth, with products priced according to the number of taxis and square miles supported.

A product line approach entails designing software for the taxi dispatch line such that portions of the functionality are duplicated from product to product. It entails building software assets that minimize the effort necessary to duplicate this functionality in software, and it entails implementing a software process that incorporates the use of these assets (and other existing software) to make new or revised products.

An analogous approach is found in the automobile manufacturing industry. Researchers at the University of Michigan studied Toyota's product development process to understand how the company makes better cars more quickly and cheaply. One supplier, Nippondenso, a world leader in radiators and alternators,

> "deliberately develops a set of designs for a set of automobiles, not a single design for a single model. That is, while developing targets for its new designs, it explicitly defines the set of automobiles in which the new product can be used. It gathers information on all prospective Toyota car models and the anticipated requirements. It then designs a product family around a single concept, producible on the same line. Nippondenso offers its customers more than 700 different alternators, providing customers with a wide variety of products while standardizing the production process; it calls this approach 'standardized variety.' For example, the development group will develop a modularized plan to standardize the various components of the alternator to meet all requirements. It might develop three different body types, nine different wire specifications, four different regulators, etc. all mutually compatible" [Ward 95].

A product line approach enables an organization to meet changing customer needs more cheaply and at a quicker pace than the competition. For example, Nippondenso offers a catalog of components that cover most of its customers' needs. Toyota chooses

from this catalog rather than contracting for a custom design primarily because the technology is advanced and because "the variety is so large and carefully designed that tailored products offer few advantages" [Ward 95].

To implement a "standardized variety" approach in software, organizations build a *production system* that can support the simultaneous development of software for multiple products. Rather than taking what was produced for a specific customer and adapting it for another customer, you expand the scope of the problem to be solved and develop common software solutions for a line of products. You develop a base of *software assets*.

Production System

A product line approach refers to a software engineering capability tuned to producing members of one or more product lines. Intrinsic in the capability are two lifecycle processes. One lifecycle focuses on the development of assets. This process is usually referred to as *domain engineering*. The other lifecycle process focuses on the development of software products. This latter process uses software assets to build related but different products. It is often referred to as *application engineering*.

In many organizations, domain engineering and application engineering begin under a single project. As organizations evolve a product line approach, the two processes often become separate functions linked to marketing.[1] Organizations evolve into what Deming refers to as production systems [Deming 93]. A production system is a way of organizing people and functions to improve continuously the production capability and core technologies of an organization.

Figure 2-1 shows Deming's production system adapted to software. The boxes represent key functions in the system. The arrows show the down-stream customers for the process of building and using software assets. In practice, of course, all functions talk to each other: marketing to application engineering, application engineering to domain engineering; and cross-functional teams develop and deploy software assets and process improvements. Management facilitates the flow of information among these functional groups. They motivate the groups to coordinate their activities and engender mutual commitment to one another's success.

Domain engineering is an upstream supplier of assets and processes to application engineering. It consists of three activities: domain analysis, domain design, and domain implementation. Domain analysis identifies the similarities and differences in software requirements across a variety of products. The output of domain design is typically robust design and component specifications. Domain implementation packages the design and component specifications into a usable form for application engineering,

1. For example, this is the organizational structure evolved at Celsius Tech [Brownsword 96] and Hewlett Packard [Rix 92]. A separate development group supplies components to customer projects in different business units.

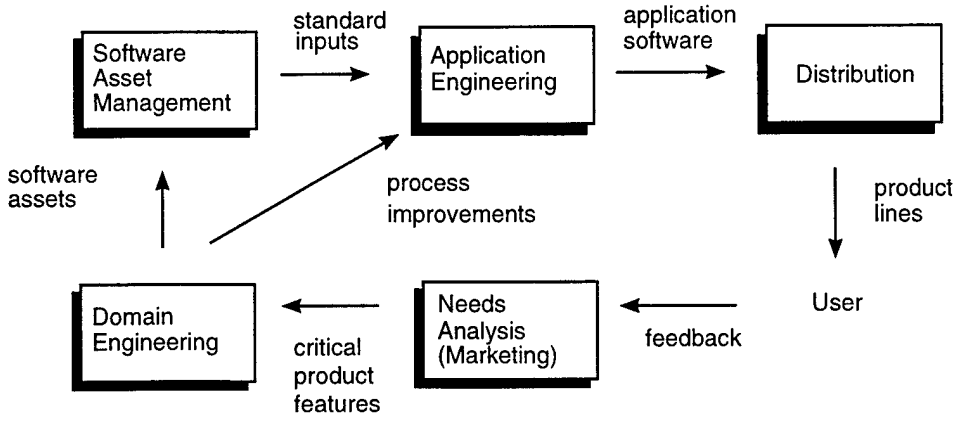


Figure 2-1: Production System

including class libraries, code generators, and design handbooks. To facilitate integrated problem solving, application engineers usually participate in domain engineering activities.

Assets

The assets supplied by domain engineering depend on the solutions common to the products in a product line. Reusing these solutions reduces or eliminates work that otherwise would be required to build each product. Typically design and programming work that only needs to be done once is codified in assets, leaving to application engineering the work that differentiates each product. Assets thus narrow the decisions that have to be made by application engineers; depending on the asset, requirements analysis, design, coding, integration, or testing tasks are simplified.

A *software asset* is a description of a solution or knowledge that application engineers use to build or modify products in a product line. To reduce work, the description must be able to explain, or implement through manipulation, changes necessary for different products. The description may be executable.

A partial solution or knowledge is embodied in an asset to make it tangible to an application engineer. For those assets that embody a partial solution, mechanisms are provided in the description so that changes can be made without full knowledge of the inner-workings of the asset. For example, a common function may be implemented as a generic software component in which specific behavior is defined via parameters. The same function may be implemented as an abstract class that is subclassed to create specific software instances. Or, for multiple changes to products, a compiler of a domain-specific language may be used to generate software code.

For those assets that embody knowledge, the description is structured so that the engineer can traverse it to find specific information about making changes. For example, an engineer may search a domain model for all the variations in behavior that were antic-

ipated in a common function at the time it was built. The same engineer may review an architecture to identify the software modules that must be changed and which design conventions to use. An engineer may browse tag fields in a product notebook to learn the physics modeled in the software.

Figure 2-2 lists many of the common assets that support different phases in the application engineering process. Prototypes of the system, domain models, or a requirements database help engineers to understand and confirm changes desired by customers for new or enhanced products. Simulation tools, notebooks, architectures, and design records help engineers understand the software as a whole as well as the portions that have to be changed. These changes are implemented in software by modifying, instantiating, generating, and/or combining software components, and then by validating the resulting system using existing test cases and data or regression testing tools.

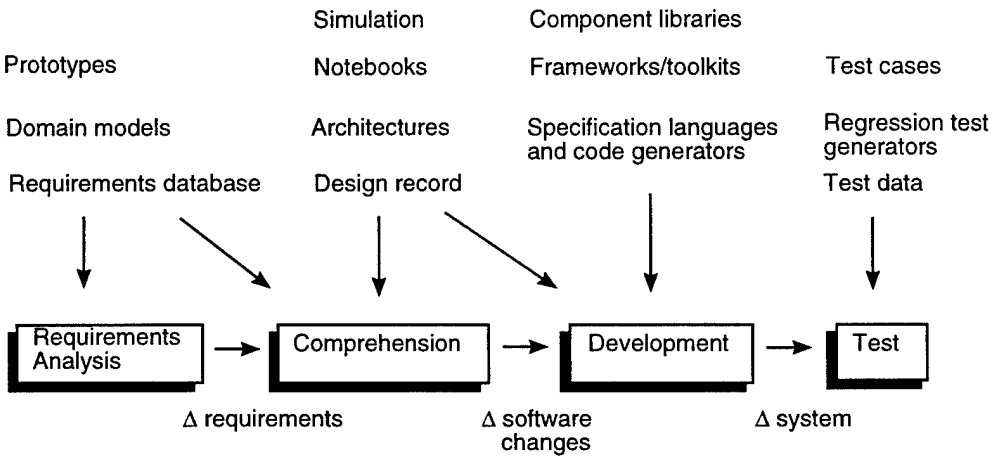


Figure 2-2: Software Assets for Different Phases of Application Engineering

Not all of these or other assets need to be built for application engineering. One objective of the investment analysis approach is to determine which assets recommended by engineering will have the greatest economic return.

Scope

In a product line approach, an organization invests in a production system and a set of assets to reduce the total costs and time-to-market for a mix of products. The number of different products in this mix is referred to as its scope. It is important to distinguish the scope of products that can be built from a set of assets from the scope of a product line. They are not necessarily the same. The economic return of an asset depends significantly on the times the asset is used to build members of a product line. But the number of different products that can be built using assets is constrained by the solution (of which the assets are a part) and not by the market. This section elaborates on this distinction in more detail. Strategic implications of this distinction are discussed in the last chapter of this report.

A *product line* is a group of products sharing a common, managed set of features that satisfy the specific needs of a selected market. These features provide a core benefit to customers in the market. Individual products in the line, however, differ along attributes that affect the buying behavior of different sets of customer in the market; for example, quality,[1] price, reputation, auxiliary features, look and feel, and distribution channels [Kolter 91].

The assortment (scope) of products in a product line is determined by market demand. Companies seldom grow by providing only one product to customers. To increase market share and revenue, companies add additional products that offer unique features for particular customers. Their goal is to define and build products that get closest to satisfying subsets of customers who, as a whole, comprise the market.

For example, recall that the product line for the car navigation market consists of products that provide information about the real-time position of a car relative to its surrounding environment and final destination. Figure 3-1 breaks the car navigation market into segments and lists some attributes that vary across these segments. Each segment represents a set of customers with a particular profile of needs and preferences. For example, taxi drivers and couriers would use car navigation devices to identify destinations and the fastest path to get there. Drivers of off-road 4x4s would use car navigation to locate rough terrain and impassible streams.

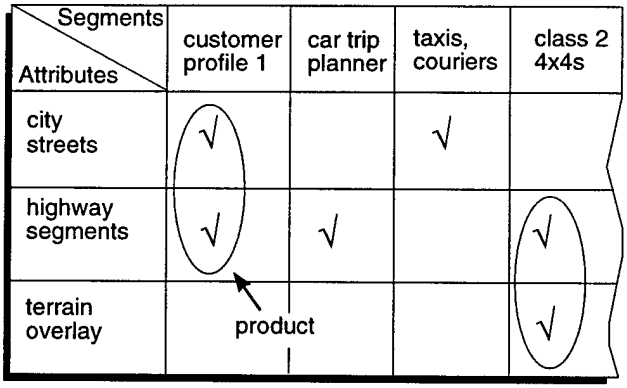


Figure 2-3: Products in Car Navigation Market

The rows show a small subset of product attributes: the different types of geographic information that are provided in different products. (Remember there are other attributes such as price and level of service.) The check marks indicate which attributes are important to a specific customer segment. A bundle of these attributes describes a product (see ovals in Figure 3-1). The scope of products sold in the car navigation market form a product line. The scope is determined by marketing, including target customer segments, distribution channels, pricing, and customer service.

1. Quality refers to the ability of the product to function as desired.

A *product family* is the group of products that can be built from a common set of assets. Products in a family typically share a design, components, and norms for system integration [Sanderson 91, Meyer 93]. The scope of the family depends on the robustness[1] of the solution unifying the assets into a functioning system, including the physics or business rules, the coordination strategy used for data and control, and the system platform. These abstractions lend stability to the structure of the software, thus enabling software components to be developed for reuse.

These solutions and the possible products that can be built are independent of market demand. In our example, both the taxi dispatch and the car navigation product lines are part of a product family centered around a digital, geographical database system that processes in real time the location of moving vehicles. Alternatively, a product line may encompass more than one product family. The scale of features is so great that, for example, a different system architecture is required to handle the higher data throughputs of the high-end products.

Thus, a product line approach involves building software assets and a production system for a product family whose scope encompasses a product line. A product line consists of products sold in a market. A product family consists of the products that can be built from a common solution and a set of assets. A production system is a way of organizing people and functions to improve continuously assets and products in product line.

2.2 Economies of Scope

Economies of scope can measure the advantage of a product line approach over an organization's current process. *Economies of scope* are the savings (or profit) that are obtained from using technology to build a greater diversity of outputs (e.g., a greater variety of widgets) with the same or less input (e.g., person hours). This contrasts to *economies of scale* where savings occur from using technology to produce a greater volume of a single output (e.g., a greater number of identical widgets) with the same or less inputs. Economies over an organization's current process result when fewer inputs are needed to product a greater variety of products.

In the manufacturing industry, economies of scope exist when one plant can produce a variety of products at a lower cost than a combination of separate plants, each producing a single product, can. A company producing both cars and trucks at one facility has lower costs than two companies, with one producing trucks, and the other producing cars [Pindyck 91]. This is because the single company can share designs, parts, and assembly techniques common to both types of products. For example, Chrysler saved

1. Robustness refers to the ability of a solution to be extended or adapted to accommodate changes in product attributes without other key attributes either being lost or becoming unpredictable. The solution has sufficient flexibility to enable it to evolve into many variants.

money by producing different car models from one design. From 1980 until the cab-forward design was introduced, all new cars in the compact, sports car, minivan, and full-size product lines were variations of the K car design. New models were part of a K car product family [Boone 89].

In the airline industry, the creation of hubs is another example [Huston 88]. When the industry was deregulated, airlines were faced with the problem of gaining market share and offering a variety of destinations from each city while minimizing the increase in the number of aircraft. Airlines discovered that by redesigning their routes to feed passengers to intermediate collection points (hubs), they could offer a greater variety of destinations with fewer aircraft. The greater variety of destinations also increased the volume of passengers to the extent that economies of scale were achieved at the hubs.

In software, economies of scope measure the capability of a production system to lower input quantities as greater variety is produced. This occurs whenever an input shared in two or more products is subadditive [Panzar 81]: that is, fewer person hours are needed if the products are produced jointly than if the products were produced independently. A software asset that can be used in a large variety of products with less effort than is currently expended exhibits economies of scope. Designing products in a line to be composed from standard parts (instead of custom-built, unique solutions) will produce economies of scope.

Economies of scale measure the capability of a technology to lower input quantities as a greater amount of one kind of output is produced. Economies of scale exist whenever increasingly smaller units of input are needed to produce another unit of output. For example, the costs of laying and operating eight-inch pipe are not much more than the costs of laying and operating four-inch pipe, but the volume of oil that can be transported is nearly 6/10 greater. A software tool that produces more lines of code with less effort exhibits economies of scale.

You cannot substitute a measure of economies of scale for a measure of economies of scope: Economies of scale and economies of scope measure two distinctly different capabilities. Producing a greater variety of software with less effort and at a faster speed does not correspond to an increase in lines of code per hour. For example, when modifying one component, it may only take a few hours to change many lines of code because the logic that checks for legal input values is repeated in various forms throughout the program. Changing another component may involve many hours of analysis, yet only require the modification of a few lines of code. Building and reusing software components may decrease the number of lines of code produced per person hour [Henderson-Sellers 93].

Diseconomies of scope occur from mismatches of assets and from poor coordination. For example, architectural mismatches occur when different assets hold conflicting assumptions about the structure of the application, its development environment, or its operating environment [Garlan 94].

Poor coordination occurs when there is no central control of a production system. Management of the software process must extend beyond a single development project and a single time period to include multiple projects over multiple years.

To illustrate the economies of scope for a production system, consider the firm mentioned earlier that sells products that provide geographical location information in digital form to users in diverse situations. You may recall that this firm has a line of products for the taxi dispatch market, and is entering the car navigation market. This firm has now expanded its product mix and now sells new systems in four product lines: house arrest, home security, police staffing, and car navigation. Figure 2-4 shows a partial list of the assets that are shared across these products. The check marks indicate the sets of products that employ the asset. The pattern of check marks indicate which assets are used the most.

Products / Assets	house arrest	home security	police staffing	car navigation
geographic database	√	√	√	√
remote control interface		√		√
locator specification language			√	√

Figure 2-4: Asset/Product Matrix

From this matrix of assets and products, you can see which assets have more value. When an asset is used in a large percentage of the products in a product line that is part of a growing market, it is more valuable than an asset that is used in a small percentage of the products in a product line that is stagnating.

When the goal is to create more variety with less effort, as is the case for software development, any investment analysis of assets must be based on economies of scope and not economies of scale. Through investment in assets, we seek to maximize what can be shared across different products. Economies of scope exist when it is cheaper to use specific combinations of software assets to produce multiple products.

2.3 Calculating Economies of Scope

Economies of scope are calculated based on opportunity costs. Building software systems with assets presents an opportunity to reduce costs and increase profits or market share. This opportunity is foregone when you choose to continue using the organiza-

tion's current process rather than invest in software assets. The net loss of benefits from a product line approach is an opportunity cost of using the current process. An opportunity cost equals the net benefits (positive or negative) from using the same resources in a rival course of action.

Opportunity costs vary across organizations, and within organizations, opportunity costs vary across development activities. For example, if in one organization, projects coordinate the development of common application services, the opportunity cost from not investing in additional assets may be negligible since the organization is already sharing inputs. However, in an organization in which projects are working in isolation without assets, the opportunity cost will be much higher.

A cost function for economies of scope is difficult to construct because the economies depend on the variation in product variety, and because software changes cannot be measured in constant units that correspond reliably to effort and hence, costs. Thus, you can construct either an input minimization function that compares quantities of inputs or a profit maximization function that compares revenue streams.

A profit maximization function determines economies of scope from profit margins rather than from input cost savings. Thus, you can consider the impact of software assets on revenues as well as on costs. For example, a production system with strong economies of scope will have a shorter product cycle time. Because less effort and consequently less time is needed to build a different product, the product reaches the market sooner. Any increase in revenues resulting from this would be included in the function.[1] In addition, assets will be chosen for their contribution to profits. Thus, an asset that costs more and is used in fewer products than another asset may still be more valuable if it is used in products having a higher probability of higher revenues.

Although the advantages of a profit maximization function have been outlined, the construction of a function is complex and still needs investigation. Because a profit function includes input costs, and because the purpose of this report is to introduce concepts for valuing software assets used in a line of products, we introduce an input function.

Equation 1 gives a general formula to calculate economies of scope based on differences in input quantities (effort) needed to perform activities that accomplish an outcome. Essentially, the total recurring costs to use M number of assets in a given V vector of products are subtracted from the total costs to produce the equivalent outcome using the current process and existing assets.

The process outcome being compared is usually an artifact, and the artifact depends on the assets. For example, if the assets are components, the artifact is the portion of system behavior implemented in software. If the asset is a simulation tool or a guidebook,

1. McKinsey and Co. developed an economic model that shows that high-technology products that reach the market six months late, but within cost will earn 33% less profit over five years; whereas meeting the market window but being 50% over cost will reduce profits by only 4% [Dumaine 89].

(1) $$S_M = \sum_{i}^{M}\sum_{j}^{V} y_{ij}^{current} w^{current} - \sum_{i}^{M}\sum_{j}^{V} y_{ij}^{assets} w_{ij}^{assets}$$

Where:

S_M = economies of scope for portfolio of assets M

y = quantity of effort

w = cost per unit of effort (hourly expenses)

M = number of assets in portfolio

V = number of planned products

the artifact may be a document of design changes. Generally the calculation involves comparing costs of the activities (including rework cycles) required to produce the artifact. Since activity cost data is not always available, the formula in Equation 1 can be adjusted somewhat to accommodate the data that is available. The above formula assumes that for the current process, the cost per unit of effort, w, is constant; whereas for the process with assets, the cost can vary from asset to asset.

Chapter 3 Construct Asset Portfolio

To maximize economies of scope as defined in Equation 1, we need to consider non-financial criteria that define the profitable sharing of solutions and knowledge. These criteria are applied to similarities that exist in software products or in software development and maintenance activities. Information is collected and analyzed in the following steps:

- characterize product variety to scope the different products for which assets will be built
- screen patterns identified in product composition or production for their potential for economies of scope
- choose assets for these patterns that are usable by application engineers and reduce current costs
- evaluate the assets as a portfolio to diversify risk and optimize economies

This chapter is divided into sections, and each section covers one of the above steps.

Under ideal conditions, a team composed of professionals from marketing, engineering, management, and finance would work together to compile the information needed to construct an asset portfolio. Different members would generate and supply information germane to their field. Marketing provides information on the variety of products and how they differ. Engineering identifies similarities in these products that, as assets, will eliminate or reduce the work of application engineers. Economies of scope come from closely coordinating software assets with the products planned in the future.

3.1 Characterize Product Variety

The objective of this step is to compile the following:

- anticipated products to be built over the next 3 years
- critical product attributes that drive sales
- likelihood of these products being sold, given economic conditions and company strategy

In this step, the team reaches consensus on the scope of products to be "mined" for similarities by engineering.

Marketing forecasts typically include the products to be sold, data on the gross margin and the dollar profit, and projections of revenue and market share. From this forecast, the team defines the *product variety* (the vector of products) for identifying software assets and calculating economies of scope.

This product variety establishes the changes in product attributes - differences in individual products - that must be accommodated by assets. It determines the scope for modeling and delimits the variation in any similarities identified. In addition, the critical attributes define engineering priorities.

Pedestal Systems Product Line

In the following chapters, we will introduce concepts and then apply them to a product line of pedestal systems. A pedestal system is an electro-mechanical system that moves detection devices and weapons as they track moving airborne objects (see Figure 3-1[1]).

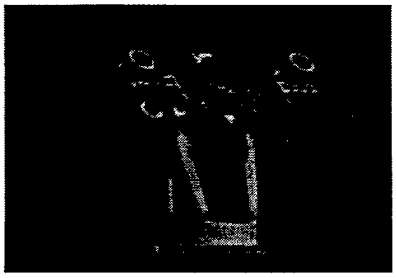


Figure 3-1: Pedestal with Optical Devices

The detection devices are normally mounted on a platform that pivots up and down while the entire pedestal moves left or right. Pedestal systems are delivered to many types of customers. Similar to most industrial product markets, the customers are identified by the types of systems they buy. Our hypothetical product line consists of four segments:

1. Air Traffic Control systems including mounts for en route radars and airport surveillance radars
2. Object Tracking systems for tracking ordinance at test ranges
3. Stationary-platform Point Defense (Stationary PD) systems such as surface-to-air missiles
4. Movable-platform Point Defense (movable PD) systems such as anti-aircraft guns on ships.

1. Photograph courtesy of Contraves, Inc. 620 Epsilon Drive, Pittsburgh, PA 15238.

Figure 3-2 shows the variety of products planned over the next three years under rosy economic conditions. Twenty-five products are anticipated, with movable point defense pedestals for the Navy accounting for more than half of total sales. Sales targets and expected revenue are not shown.

Air Traffic Control	Object Tracking		Stationary Point Defense	Movable Point Defense	
en route	test sites	surveillance	missiles	gunnery	missiles
4	2	2	3	10	4

Figure 3-2: Rosy Market Forecast

Figure 3-3 shows the products forecasted under dismal economic conditions.

Air Traffic Control	Object Tracking	Movable Point Defense	
3	2	2	3

Figure 3-3: Dismal Market Forecast

The likelihood of either scenario occurring appears to be 50%.

There is a lot of uncertainty in a marketing forecast. To reach consensus on product variety, the team chooses to have engineering look for similarities given the full variety in the rosy scenario, and to begin with the products identified in the dismal scenario. They also decide to estimate and calculate the economies of scope for each scenario based on what engineering identifies.

Figure 3-4 shows some of the key features that differentiate products in the hypothetical product line. First, the payload mounted on pedestals varies from product to product. A pedestal may support any or a combination of the following detection devices: optical, video, laser, electronic warfare (EW), forward-looking infrared (FLIR), and radar.

The agility of the objects tracked by the pedestals also varies. For example, civilian aircraft have a less volatile flight profile than do military aircraft, which is why agility is not as an important sales point for air traffic control customers as it is for military ones.

These features affect how the software behaves. For example, gunnery systems have vibration that must be compensated for in pedestal tracking algorithms. The critical ones are shared with engineering.

Segments / Attributes	Air Traffic Control	Object Tracking	Stationary PD	Movable PD
payload diversity	radar	all six	radar	radar EW optical
target agility	low	high	high	high
platform stabilization	no	no	no	yes

Figure 3-4: Attribute/Segment Matrix

3.2 Screen Patterns

Products in a product line have many similarities that can be leveraged to reduce the amount of work in software development and maintenance. These similarities manifest themselves as *patterns*, either as partial solutions that are replicated from product to product (such as an algorithm or a set of operations), or as knowledge that is reapplied in building each product (such as requirements or test data).[1]

Patterns are readily identified or synthesized through modeling the software. Using a software analysis or design method, the team develops a first-pass description of the software system that is common to the scope of products established in the previous step. This description helps the team infer the product-specific knowledge, processes, and partial solutions that may be similar in the product line.

The team (or engineering) need not fully flesh out the model to identify candidate patterns. Often a block diagram and an analysis of the variability of the blocks in the diagram is sufficient. The objective is to identify the possibilities for economies of scope and the technical uncertainties that need to be resolved. For example, a system description may indicate a common set of services or a common user interface.

Conducting a high-level domain analysis is recommended. Most modeling methods are designed to describe the software system for one product. *Domain analysis,* on the other hand, is a process that develops a description of the entities, operations, and relationships in a real-world system that is implemented, at least in part, in a family of software

1. The American Heritage Dictionary defines a pattern as "a consistent, characteristic form, style, or method." This is the spirit in which the term is used in this report. Our use of the word expands the concept, yet is consistent with that described in some object-oriented literature. Gamma concentrates on patterns that solve design problems [Gamma 94], while we concentrate on patterns that exist in the composition of multiple products or in the production of multiple products. The objective is to find similarities that reduce engineering tasks and decisions.

products. The description documents the variations in the system that may occur over time and across these products. To identify patterns, we look for cohesive clusters whose variation is well-defined and well-contained.

The view chosen in software modeling implicitly determines the similarities that emerge. Different views emphasize different properties and structures in a software description. Details captured in some views are ignored in others. For example, a view that focuses on the runtime behavior of the software will reveal patterns in the coordination and communication among executing processes. A view that describes the user interactions with the system may suggest common patterns (motifs) for software operation. As the view determines the content of the description, it circumscribes the patterns that may be identified.

Typical views that are useful include the following [Clements 96]:

- the conceptual (logical) view: This view describes the interaction of entities in the problem domain. This view focuses on requirements - key application functions - and suppresses implementation details.
- the modular (development) view: This view describes the static organization of the software: the software components and their connections.

The view chosen in modeling also indirectly dictates the expertise that will be required of application engineers that use the assets. For example, engineers that develop software for household appliances typically express the behavior of these appliances using system state tables. A view that describes the system resources (CPUs, networks, etc.) may identify reusable parts, but not ones these engineers could modify without retraining. You want to choose a view that is compatible with the skills and domain expertise of application engineers.

Design and scoping decisions are often made while modeling, and this also creates or limits recognizable patterns.

Criteria

To maximize the potential for economies of scope, engineers need to be cognizant of the following criteria while modeling a software system to synthesize patterns: scope, span, stability, and encapsulation.

Scope refers to the ubiquity of a pattern - its existence in the variety of products to be built.

Span refers to the breadth of the system included in a pattern. It is the portion of a system description covered by a pattern. For example, in the description of common system services, the span of a pattern can vary from a small routine that provides one service, to a runtime kernel that provides multiple services.

Stability refers to the soundness of a pattern - the extent to which it is cohesive and well-defined across product variety.

Encapsulation refers to the extent to which changes are contained within the pattern and do not affect other parts of the description.

These criteria can also be applied to the knowledge or processes that are reused from product to product, such as methods or common mathematical transforms.

Kiviat Diagrams

Patterns can be compared against these criteria using Kiviat diagrams. *Kiviat diagrams* are graphs with an axis for each criterion. The axes share a common origin. A diagram is made for each pattern, and values are plotted on each axis. In general, the greater the area of the polygon created by line segments connecting the values on the different axes, the more likely the pattern will create economies of scope. These diagrams greatly help teams screen patterns.

To construct kiviat diagrams, you have to define a scale for each criterion in terms of the description you used for modeling. You can use Figure 3-5 as a guide. This table creates a scale from 0 to 1 for each criterion, where 1 is the most desirable.

Dimension	Description	Benefit	Operational Definition
Scope	Proportion of product variety having pattern	Pattern is used in all products.	Number of products having pattern divided by total number of products
Span	Proportion of system description encapsulated in pattern	Greater amount of work is avoided.	Number of items in pattern divided by total number of items in description
Stability	Description of pattern is well-defined and cohesive. Description does not vary across products.	Complexity is reduced and change is bounded.	Number of items in pattern less conditional items, that quantity divided by total number of items in pattern
Encapsulation	Changes are local to pattern, and not propagated outside of pattern.	Costly change dependencies are minimized.	Number of changes encapsulated in pattern divided by total number of possible changes to pattern

Figure 3-5: Pattern Criteria

The goal is to identify a set of patterns that together have the largest scope and span and whose variation is below the threshold that can be accommodated by the expertise of application engineers and by the differentiability of the assets. Engineers will want to identify the level of 'sameness' whereby differences across product variety are manageable and cheaper to implement using assets than without assets.

Pedestal Patterns

For the pedestal product line, the team applies an object-oriented analysis technique to identify patterns. Two scenarios describing the operations of two products are developed and scripted in a notation.

The first product is a manned pedestal on a moving platform with an infrared detection device. The operator visually acquires the target using a guide scope, and then activates the software to start tracking using infrared images. The product is used to monitor delivery of naval ordinance.

The second product is an unmanned pedestal on a stationary platform with electronic warfare (EW) and video detection devices. The product is used to monitor activity along a national border: The electronic warfare device is used to track airborne objects, while the video is used to record data about the object. The system is initially rotating to acquire objects. Once an object is detected via an EW device, the system starts tracking based on EW and keeps the video recorder aimed at the object. Using EW techniques for tracking allows objects to be covertly acquired in all weather conditions at a greater distances than possible using optical devices.

The scripts are combined in a database, and common objects are identified. The team decides on two patterns: a target tracker and a servo movement controller.

The target tracker estimates the position of a target based on its previous position, compares this calculated target position with image data, and then records the actual position of the located target for the next iteration. The changes to the target tracker depend on where the payloads are mounted on the pedestal, the agility of the target, and the presence of electronic warfare information.

The servo movement controller calculates and sends instructions to the hardware that controls pedestal movement. Changes to the controller depend on the payload and needed tracking accuracy.

Figure 3-6 shows the Kiviat diagram for the two pedestal patterns.

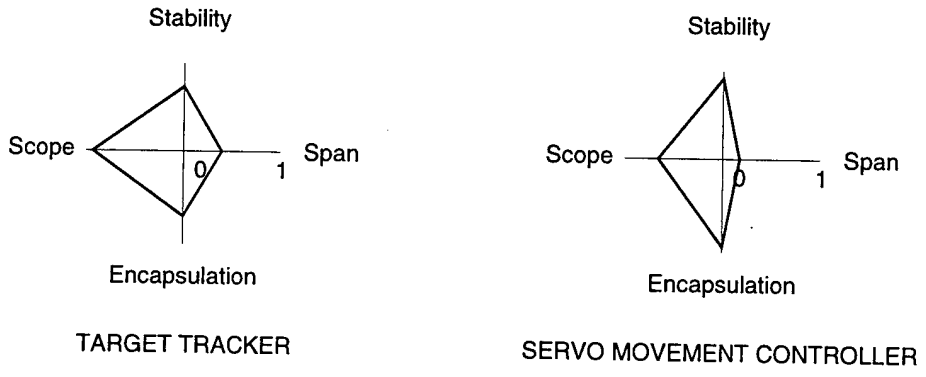


Figure 3-6: Kiviat Diagrams for Pedestal Patterns

3.3 Select Assets

Next, the team proposes software assets for the patterns. For any given pattern, the team has several choices. For example, a set of functional requirements may be realized as a suite of test cases or as a compiler for a domain-specific language, or as a framework. The choice of assets depends on the activities that contribute the most to the costs of software development and maintenance, on the flexibility required to accommodate the variety of products, and on the skill set of application engineers.

Activity Costs

A recurring activity that incurs the greatest proportion of costs is a good target for cost reduction via software assets. Maintenance data is an appropriate source for identifying those activities. In both application engineering and maintenance, activities range from making simple upgrades to extending functionality to meet new needs. In both processes, engineers are working with preexisting development artifacts. Application engineering differs in that the change activity is planned. Engineers upgrade or build new products using assets that accommodate changes in product attributes.

Schach decomposed software maintenance into seven repeated activities and reported the relative costs of each activity [Schach 94]. The most costly maintenance activities are shown in Figure 3-7. They are ordered by impact on total costs. The other

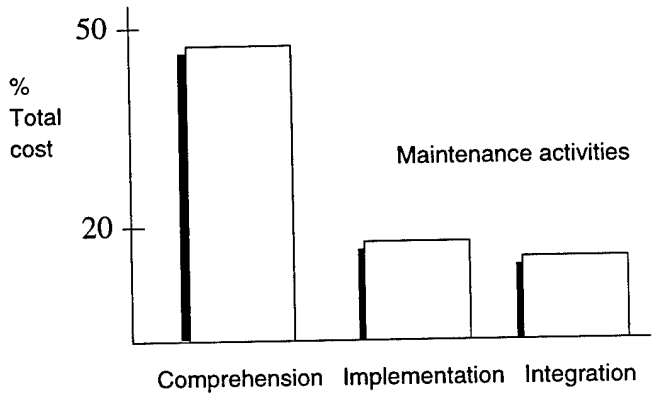


Figure 3-7: Pareto Graph of Activity Costs

four activities, requirements, specification, design, and revalidation, account for the remaining 20% of costs. According to Schach's data you would want to build assets, such as guidebooks, that reduce the time required for comprehension.

Greater economies will also occur if the asset replaces difficult work; for example, recurring patterns that are complex to redevelop.

Asset Flexibility

Asset flexibility can be defined as the number of product attributes and their values that an asset can accommodate for a given product variety.

Some assets are more flexible than others. For example, a generic module can support only a few product attribute changes, and only when the interaction among them is simple, whereas a specification language can support a greater number of attribute changes whose interactions are complex.

Depending on your objectives, you choose to constrain or augment the flexibility required of an asset. In Figure 3-8, the areas circumscribed by the lines show two possible flexibility requirements for a candidate asset. To account for uncertainty in a variety of products, you may want an asset that is more flexible: one that accommodates all attributes to point 1 (areas A and B). To reduce complexity, you may choose to constrain the flexibility to accommodate only the most frequent attribute changes (point 2).

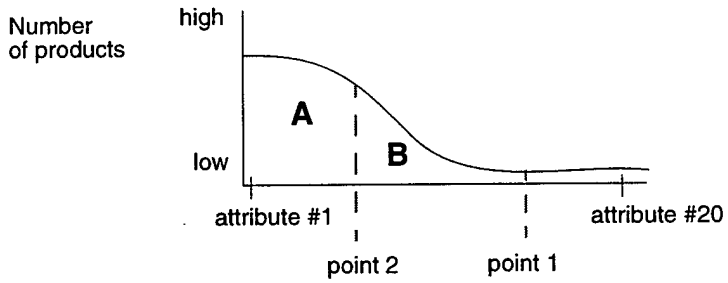


Figure 3-8: Flexibility

Robustness is a measure of the extent to which varying product attributes are supported by an asset, either by modification (if a component or tool) or by explanation (if an information artifact). If the product attributes that must change do not affect the asset, then it is robust: It is independent of the changes. If the asset is designed to accommodate all required changes, then the asset is also robust. If the asset does not accommodate a portion of attribute variety, then robustness falls quickly (Figure3-9). It becomes useless (if an information artifact) or expensive to modify (if a component or a tool). If an engineer must change more than 20% of a component, it is more cost effective to rebuild the module than to adapt it.[1]

1. Boehm reports that the cost of modifying 20% of a code component is equivalent to an estimated 90% of the cost to develop the component from scratch [Mili 95]. Thus, for reuse to be cost effective, changes must be pre-planned.

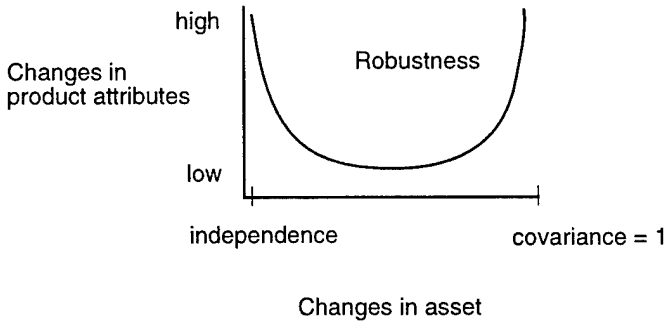


Figure 3-9: Robustness

Skill

Skill level refers to the kind and depth of expertise required to use an asset in application engineering.

Different assets require the use of different application engineering skills. In choosing assets, particularly ones that automate decisions, you have to make a conscious tradeoff about where system, technical, and product knowledge reside — in the asset or in the skills of application engineers.

You target the skill level required to use the asset according to the domain knowledge held by application engineers. For example, if your system engineers typically specify application behavior using state machines, then an application generator that accepts state transitions as input would be an appropriate asset to consider for a pattern.

The level of skill is also related to the complexity of an asset: that is, the amount of information an engineer needs and number of decisions that the engineer has to make to create an individual product. Assets that accommodate a greater variety of attribute values are typically more complex. Thus potential economies gained from an asset with high span and flexibility may be offset by the application engineering costs to tailor the asset. Using the asset may require more skills and time than is economically feasible.

Pedestal Assets

Currently, the effort to develop the equivalent functionality averages 21 person-weeks for the target tracker and 17 person-weeks for the servo movement calculator patterns. Since cost data shows that the time to comprehend and modify algorithms in the target tracker is a major cost in product modification, Figure 3-10 shows the proposals the team made for the target tracker and servo movement controller patterns.

Pattern \ Asset	1	2	3
Target tracker	Generic module: capabilities of target passed as parameters to same algorithm	Notebook containing state estimator and coordinate transformation math models	Class of objects: different algorithms, same interface
Servo movement controller	Standard module: same precision and torque, same motor commands	Guidebook discussing precision, torque, and control capabilities of mount	Application generator

Figure 3-10: Pattern/Asset Matrix

Selecting one of the three proposals shown in Figure 3-10 depends in part on the flexibility required of the asset. If the variation in attributes is so uncertain and complex that it cannot be represented in a software description, then a notebook or guidebook may be the most appropriate asset. On the other hand, if the variation is well-understood and limited, then a generic module may be sufficiently robust.

The team decides that implementing a class of objects (option 3) for the target tracker would be the best choice. The company's engineers are well versed in C++, and engineering feels the following design assumptions will reduce the variability to an acceptable level:

- All devices will be mounted in one of three predefined stations on the pedestal. The target tracker will be initialized with the coordinates of the device's position.
- Electronic warfare changes will not be designed in an asset. Because the customizations to the comparison algorithms in the target locator are unique, they will be developed by hand.
- One Kalman filter form will be used for both high and low agility targets.

The team decides to build a standard module (option 1) for the servo movement controller for the following reasons:

- The servo movement controller is tightly coupled to the choice of hardware because the hardware/software interface is complex.
- Only two hardware systems are used, one for heavy payloads and another for light payloads. (Because of costs, the heavy duty servomotor is not used with light payloads.)

Because the number of pedestals with heavy payloads is far greater than the number with light payloads, the module would be designed for reuse in heavy payload systems.

Figures 3-11 and 3-12 summarize the scope of the two assets given the engineering assertions made by the team and the forecasts made by marketing. Each figure shows the number of *different* pedestal systems that will use each asset. In the columns, the systems are grouped by market segment. Each segment is further subdivided by important attributes that affect asset use. For example, note that the servo movement controller asset is not used in pedestals with optical devices: Optical devices are light payloads. Also observe that the team assumes that EW additions or changes will be made to the target tracker asset; the target tracker will not be redeveloped from scratch.

	Air Traffic Control	Object Tracking		Stationary Point Defense	Movable Point Defense		
	en route	test sites	surveillance	missiles	gunnery		missiles
	radar	optical & light	EW & radar	radar	optical & light	EW & manned	radar
Target tracker	4	2	2	3	4	6	4
Servo	4	0	2	3	0	6	4

Figure 3-11: Asset/Product Matrix for Rosy Market

	Air Traffic Control	Object Tracking	Movable Point Defense	
	en route	test sites	gunnery	
	radar	optical & light	optical & light	EW & manned
Target tracker	3	2	2	3
Servo	3	0	0	3

Figure 3-12: Asset/Product Matrix for Dismal Market

3.4 Evaluate Portfolio

The following criteria can be used to evaluate candidate assets for the portfolio (see Figure 3-13). All but the last two have been introduced and used in the process up to now. Additional criteria can be defined.

Criteria	Operational Definition
Span	% of system covered by the asset
Scope	% of potential products using asset (asset/product matrix)
Impact	% of total software effort replaced by asset
Robustness	covariance of product attributes to asset flexibility
Skill	kind and depth of expertise required to use asset
Development Time	calendar time required to build asset
Development Costs	non-recurring costs to build asset

Figure 3-13: Portfolio Criteria

These criteria are used to design a combination of assets that, as a group, optimize the potential for achieving economies of scope. When you choose an asset, you make a tradeoff between two or more factors. For example, you may have to reduce the scope of a candidate asset to increase its robustness.

In constructing the portfolio, the goal is to diversify these tradeoffs and in the aggregate, achieve the strategy desired by the team. For example, the team may want to diversify market risk and construct a portfolio that maximizes the number of products that use the assets. Or instead, the team may want to diversify the application engineering activities that use assets and construct a portfolio that varies the kinds of assets that are developed. Or they may wish to maximize the number of assets for a subset of the product variety.

Simple graphs help teams analyze the candidate assets. Figure 3-14 show 3 graphs that compare the properties of two assets. The size of the circle represents the relative size of the budget required to build the asset. The number in the circle refers to a particular candidate.

For example, an asset with a larger span has greater potential to produce savings than an asset with a smaller span. However, as Graph (a) in Figure 3-14 shows, an asset with a large span may be used in fewer planned products. Thus the total savings may not be significant.

As Graph (b) shows, you may want application engineers with higher skills than originally planned so that you can use them to build more products.

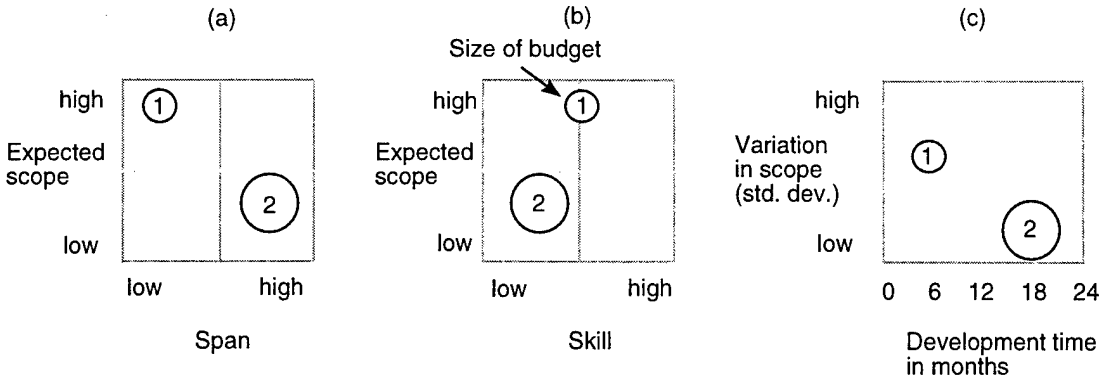


Figure 3-14: Portfolio Comparisons

Because of uncertainty in product variety, the scope of an asset can vary. For two contrasting market forecasts, one asset may be used 20 times or only 5 times, whereas another asset may be used 10 times in both cases (Graph (c) in Figure 3-14). Is that risk acceptable?

Although now we are constructing the portfolio using predominantly non-financial criteria, after modeling the investment, we can always come back to this step and reconsider its composition as part of a sensitivity analysis. In sensitivity analysis, you test how much the investment is influenced by key decisions you made in constructing the portfolio (e.g., the choice of assets, the scope of products).

Chapter 4 Model Investment

This chapter introduces techniques for modeling the investment in an asset portfolio so managers can decide whether or not to invest immediately. This chapter introduces the concept of net present value and how it is adapted for modeling investment in software assets. Decision trees are introduced to the model the different risks, and then the investment in pedestal assets is analyzed.

4.1 Net Present Value

The Net Present Value (NPV) formula is most often used to model and evaluate an investment. You model the decision to invest in a portfolio of assets as a project with cash outflows and inflows occurring over a period of time. Cash outflows are typically the expenses required to acquire an asset. Cash inflows include the cost savings and profits resulting from using the asset.

Equation 2 shows the generic form of this formula [Brealey 91]. The opportunity cost of capital, r, represents the return expected from investing in securities that have the same risk as the project. The net cash flow for a given time period is discounted to today's dollars by this opportunity cost. This discounted cash flow is the present value of the investment.

(2)
$$NPV = \sum_{t=0}^{n} \frac{C_{i_t} - C_{o_t}}{(1+r)^t}$$

Where:

C_i = cash inflows during period t

C_o = cash outflows for period t

r = opportunity cost of capital

t = time period of constant duration, typically one year

n = number of periods in planning horizon

However, this formula does not account for uncertain events that may happen during the investment and for managers' flexibility to respond to them. It assumes that managers are passive, and that they do not have the option to make intermediate decisions

to continue, discontinue, or alter the investment. It also assumes that the investment risk remains constant over the planning horizon, when actually risks change as managers make decisions. (This formula was originally invented to evaluate the return on bonds.)

4.2 Uncertainty

An uncertainty is an event that can happen, but the probability of its occurrence is unknown. A risk is an event whose occurrence has some known probability distribution. Teams modeling an investment typically convert uncertainties to risks by assigning probabilities to events. These probabilities are typically consensual, but subjective, based on beliefs, experience, analogy, and incomplete information. Part of the modeling exercise is to tease out events that can significantly affect the success of an investment, and then include them in the analysis of whether to proceed with an investment, collect more information, or abandon the effort altogether.

There are two kinds of uncertainty: technical uncertainty and economic uncertainty [Guimarães 96]. *Technical uncertainty* refers to the unknowns involved in completing an investment. Typical unknowns are the actual time and effort required to build an asset, whether the asset will perform as planned, or whether the asset will be adopted by others. Technical uncertainty can only be resolved by undertaking the investment project. The results (costs, performance, adoption) are only known once the investment is completed [Dixit 94].

Economic uncertainty refers to the unexpected events beyond the direct control of the organization, such as changes in short-term interest rates, an economic recession, and government shutdowns. This uncertainty is exogenous to the decision process - the national economy influences your decision to build assets and not the other way around. Technical uncertainty, on the other hand, is endogenous to the decision process. The decisions you make and the resources you provide to complete an investment may reduce or increase the uncertainty.

Thus, in the face of these uncertainties, management has several options:

- the option to make follow-on investments in a project
- the option to abandon the project
- the option to wait until favorable market conditions exist before investing

These uncertainties and options are included in the investment model by building decision trees.

4.3 Decision Tree

Decision trees are used to model investments when uncertainty is involved. They organize the information relevant to making an investment decision.

A decision tree is constructed by dividing an investment into a series of steps or milestones. Some milestones are management decision points: Managers have the choice to continue, defer, switch, or abandon their investment in a portfolio of assets. Other milestones represent probable outcomes from an investment decision. They reveal information that managers use to make subsequent decisions. Milestones are modeled as a node with one or more branches. Each branch represents a probable event that leads to another decision or outcome. The terminal branches usually represent the benefits created by the asset.

Decision trees describe the technical and economic uncertainties associated with cash inflows and outflows. They are a convenient way to summarize the decisions and consequences that affect cash flows.

Cash inflows are the reductions in costs or the increases in profits stemming from economies of scope. Cash inflows are the benefits that come from using an asset during a time period

Cash outflows are the expenses paid to achieve the benefits of the investment. These expenses include asset development costs and changes to an organization's infrastructure. Cash outflows often provide essential information for the next stage of investment.

The specific cash outflows to include in a model depend on the assets to be built. For example, if the assets are software components, a software architecture must be developed to avoid component mismatches. The architecture partitions the system into common components, defines how these components interact, and specifies how the components are assembled in a working system.

If the asset to be built is an architecture, an analysis of the enabling technologies and requirements for products in the product line should be a prerequisite. The Software Engineering Laboratory (SEL) at the National Aeronautics and Space Administration's Goddard Space Flight Center found it necessary to conduct a domain analysis to increase the scope of systems covered by an architecture [Stark 93].

A *deployment strategy* is a plan of action for phasing in a portfolio of assets. The process of developing and deploying a portfolio is divided into a sequence of steps that mitigate undesirable events (uncertainties) and capitalize on desirable ones. This sequence of steps constitutes a deployment strategy. The strategy defines the management decision points in a decision tree.

Depending on the uncertainties involved in building and deploying assets, and ultimately in making the transition to a production system, different deployment strategies are developed. For example, if organization adoption is uncertain, you may want to build

and deploy assets incrementally in a showcase project. Policies, plans, practices, and reward measures would be developed by a change agent as part of the effort to integrate the first asset in the project's process. Based on the success of this effort, subsequent assets would be adopted with less expense. Training would be developed.

If the technology is uncertain, you may wish to pilot an asset in a demonstration project before including it in a critical project. Perhaps before the asset is built, a team would evolve the design over the course of two product deliveries. Not until demand for a line of products begins to grow quickly do you build the asset.

4.4 Investment Decision

The uncertainties in the tree affect how you make the decision of whether to proceed today with the investment. Investment decisions are defined by decision rules. For example, when using the NPV formula given by Equation 1, the decision rule is as follows:

- if the NPV is positive, proceed with the investment
- if the NPV is negative, do not proceed.

However, because of uncertainties, this rule does not always hold. In the case of technical uncertainty, a negative NPV may still warrant immediate investment. Because the calculation sums all possibilities of cash flows, it hides the value of the information gained after the first step. To proceed with the investment, you only want to know if the possibility of positive payoff is greater than the expense of the first step. Thus, the calculation is made by pruning all downside branches in the tree, and you proceed if this modified NPV calculation is positive [Dixit 94].

Technical uncertainty increases the willingness to invest by raising the expected value of the investment and reducing the impact of the overall cost. We assume that a rational manager will not proceed to the next step unless there is good news.

On the other hand, economic uncertainty makes it less attractive to invest now. An investment with a positive NPV may still be uneconomical if the variation and impact of an event is sufficiently large. Although the uncertainty increases the value of the investment (greater profits are possible), it decreases the willingness to invest (the downside loss is also greater). This variation creates a value in waiting for new information before proceeding with the investment.

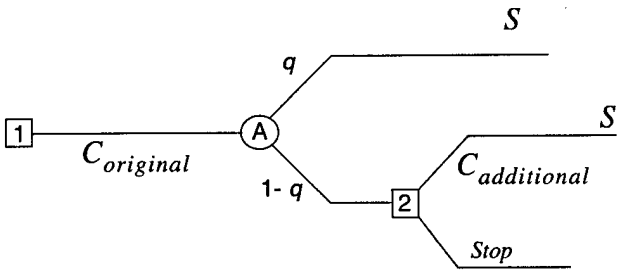
Economic uncertainty is modeled as a risk that affects investment outcomes and is usually added to the interest rate used to discount future cash flows to today's dollars. The interest rate plus risk is called the discount rate. The discount rate is tied to the opportunity cost of using the cash outflows in another investment. If the investment is risk-free (that is, the value of the investment is independent of changes in the economy) then the discount rate equals the opportunity cost of investing in safe securities: for example,

the interest rate of three year Treasury Bills. However, if the outcome is dependent on variation in the market, then it is appropriate to discount the cash flows to the opportunity cost of capital for the organization. This is the risk premium added to the risk-free interest rate that the stock market places on the organization's stock. The spread (variation) in the total return of the stock (growth in value plus dividend) over time reflects investors' estimation of the prospective revenues of the company. If the investment is a new area of business, then the discount rate should be based on the variation in return of companies having the same risk as the investment [Brealey 91].

Because of this uncertainty, with a barely positive NPV, it is wise to test whether the investment should be made now or should be postponed. You construct an alternative deployment strategy, where the entire investment or part of the investment is deferred for a period of time, and calculate its NPV. You should pursue the strategy for which the NPV is greater.

Example

Figure 4-1[1] shows the tree for a two-step project with technical uncertainty. The second step is an option because you may not undertake it.



Where:

C = cost to build asset

S = savings from using assets (economies of scope)

q = likelihood of completion

Figure 4-1: Decision Tree with Technical Uncertainty

1. For simplicity, the figure and equations do not show how the variables are discounted by the opportunity cost of capital. If the risks in developing the asset are different from those using the assets, the variable *C* would be discounted using a different discount rate than *S*.

The original cash spent on the asset creates an *option* to achieve economies of scope [Myers 84]. At point A in Figure 4-1, you will know which branch is closest to reality. If the asset is not completed, you will either exercise the option (step 2) to spend more money or cut short your losses by discontinuing the project. You will base your decision on the estimated payback (S) at the time.

The decision is whether you should invest now knowing that there is a q probability of obtaining S savings. Your investment decision is shown in Equation 3.

(3) $$NPV_1 = -C + qS + (1-q)0$$

The downside branch is pruned since we can abandon the investment if a second phase of additional cost is required. This NPV is greater than the NPV that would be calculated using the traditional net present value calculation (See Equation 4).

(4) $$NPV_A = -C + qS + (1-q)(-C_{additional} + S)$$

Decision trees can become complex. However, the purpose of decision trees is to model the links between the most important steps in a project; not to show the full range of possibilities in cash flows. You identify the primary consequences of a series of investments. Then, assuming that you will minimize failure, you evaluate the investment.

4.5 Pedestal Asset Portfolio

In this section, we model the investment in a portfolio consisting of two pedestal assets. We have established the variety of products planned to be built in the future, identified the patterns common to this product set, and proposed assets that are technically viable. With this information, we can develop a simple decision tree to calculate the value of the investment.

Deployment Strategy

The deployment strategy for the hypothetical pedestal assets is simple.

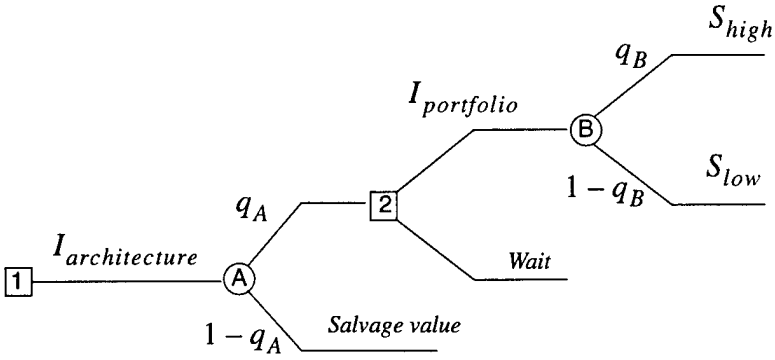
Because the technology is well understood and projects currently coordinate and share common work such as math modules, there is not much uncertainty in the projects adopting two more components. However, pedestal systems are not currently built using a common architecture. This will have to be developed first.

Developing a software architecture requires a major cash outflow that precedes the expense of developing assets. This architecture is necessary because it defines the overall structure of the software, and thus specifies how specific products will be composed using the components. Key decisions affecting component design would also be made. High-level design issues such as the flow of control, the storage and transportation of data, and the design of hardware interfaces would be resolved.

Also, economic conditions may have changed after the architecture is built. Management may wish to plan for different actions depending on different market scenarios.

Decision Tree

Based on our pedestal deployment strategy, the general decision tree for analyzing the asset portfolio is shown in Figure 4-2. Depending on assumptions made in estimating, different values can be substituted for the variables I, q_A, q_B, S_{high}, S_{low}, A, and B.



Where:

$I_{architecture}$ = investment expense for architecture

A,B = uncertainties corresponding to points in time

q_A = likelihood that architecture meets requirements

$I_{portfolio}$ = investment expense for asset portfolio

q_B = likelihood that market is good

S_{high} = economies of scope from components given a good market

S_{low} = economies of scope from components given a bad market

Figure 4-2: Decision Tree for Pedestal Asset Portfolio

At step 2, you have the option to invest in the portfolio or stop the project. The decision is based on an evaluation of the architecture and the market. If the architecture meets requirements, you will consider building the assets. If the architecture fails to meet requirements, (e.g., establishing standard interfaces for the planned components) you will abandon the project and the architecture will have little to no salvage value.[1] Before al-

1. "Salvage value" is the term used in finance to describe the benefits that can be recovered when the project is "scuttled." Although knowledge was gained by building an architecture, the investment is a sunk cost. So, in this scenario, the "salvage value" is zero.

locating resources to build the assets, you will also consider the market. If the market has weakened significantly, you may want to postpone investing in the portfolio, and use the architecture on the few products that are developed.

All cash flows in Figure 4-2 are discounted to dollars at time t, where t is either the time at step 2 or step 1. Discounting cash flows at a continuously compounded rate r for n years involves multiplying the cash flow by e^{-rn} . Since the technical risk of building the architecture is modeled in the tree, and is independent of economic uncertainty, we will discount the cash outflows by the "risk-free" interest rate.

We assume that the risk of return for the pedestal product line is the same as the risk of return for today's products. The asset portfolio diversifies downside technical risk, and the advantage of time-to-market in the defense market is unclear. Also, since the risk of return for the pedestal product line is the same in either market scenario, the cash inflows will be discounted at the company's current opportunity cost of capital.

Although the cash savings (or cost avoidance) from economies of scope S begin when the first product uses the assets (just after milestone B), the savings are discounted to today's dollars, assuming they are only realized at the end of the period included in the economies of scope estimation. Since the economies depend on all products being developed, this assumption is reasonable.[1] But because cash inflows in year three, for example, are worth less in today's dollars than cash inflows occurring in year two, this assumption errs in making the assets less valuable than they really are.

Cash Outflows

The decision tree shows two major sources of cash outflows: the expenses of building an architecture and two assets. Based on multi-year company experience and the detailed modeling conducted by the team, it is estimated that the pedestal software architecture would take four people six months, or about 96 person-weeks, to develop and test. Pedestal systems have not been that large in size, and being an embedded feedback control system whose basic functionality is constant across products, a cyclic executive architectural style has worked well.

We estimate the effort to build the target tracker and servo movement controller assets will take 2.25 times the effort to implement the equivalent functionality in a current product.[2] Considerable time will be spent eliciting requirements from all stakeholders and documenting assumptions and mathematical derivations embodied in the assets.

1. Also, it is very difficult to predict with any level of confidence when each product is sold after milestone B, much less to predict the changes that would be made in each product. This is another reason why a cost function is difficult to construct.

2. The effort multiplier recommended for estimating the budget of a reusable component varies. Mili notes that multipliers range from 1.5 to more than 2 [Mili 95]. Boehm reports that in AT&T's experience, a factor of 2.25 is appropriate. Because greater reliability is needed, Boehm recommends multipliers in this range [Boehm 95].

Currently, the effort to develop the equivalent functionality averages 21 person-weeks for the target tracker and 17 person-weeks for the servo. Thus, the investment expense for the target tracker and the servo movement calculator assets are 47 and 38 person-weeks, respectively. The assets will be developed over six months.

Cash Inflows

Cash inflows are the savings from economies of scope. To estimate the economies of scope for the pedestal assets, the following must first be determined:

- the average recurring effort to replicate the tracker and servo functions using the current process
- the average recurring effort to use the assets

Then for each market scenario, using the asset/product matrix to identify which products will be built using the assets, the economies of scope are calculated.

The average recurring effort for replicating the functionality of the target tracker and the servo movement controller in a new pedestal system is respectively 21 and 17 person-weeks. The effort requires senior engineers averaging an hourly rate of $37 per hour.

A learning curve coefficient could be added to these estimates to improve accuracy. With each pedestal system, engineers learn more about building products in the product line. Consequently, less effort is needed to produce each subsequent system. Over 3 years and 18 products, the current process cost may decrease significantly. To account for this learning, effort estimates can be multiplied by a coefficient that decreases in value with each product development.

The recurring effort of using the target tracker asset in application engineering depends on the modifications that would be made from product to product. Based on the design assertions made in the previous chapter, Figure 4-3 gives low and high effort estimates for an abstract class designed to accommodate device changes. For non-electronic warfare devices, recurring effort for minor changes totals 1.25 person weeks. For major changes,[1] the recurring effort equals 3.5 person-weeks. For electronic warfare (EW) de-

	Non - EW tracking devices	EW tracking devices
Minor changes	1.25	1.75
Major changes	3.5	8.5
Average	2.4	5.1

Figure 4-3: Estimated Effort to Use Target Tracker Asset (in Person-Weeks)

vices, recurring effort totals 1.75 person-weeks for minor changes, and 8.5 person-weeks for major changes. The averages are shown in the figure.

The estimated recurring effort for using the servo module ranges from 1 to 3 person-weeks, for an average of 1.5 weeks.

Because the assets require fewer skills to modify, fewer senior engineers can be assigned the task. The hourly expense therefore, is anticipated to average $30 per hour.

Because an architecture and assets may suffer from entropy, these may need to be modified over time. Over time, changes in the assets tend to increase complexity and reduce effectiveness; more effort will be required to use the assets. Accordingly, recurring effort estimates may be multiplied by a coefficient that increases slightly with each product development.

Asset/product matrices are used to determine which of the above effort estimates should be used to calculate the economies of scope. For some of the object tracking and movable point defense products, the effort estimate for EW tracking devices is used for the target tracking asset. The servo movement controller asset is not used in pedestals with optical devices.

By totaling the difference in cost to build each product with and without assets,[1] we arrive at the economies of scope. Figure 4-4 and 4-5 summarize the figures used to calculate economies of scope for rosy and dismal market forecasts, respectively. Because assets will not be available until the second year, only the number of products forecasted for years two and three are used.

	Number of products, *V*	Current effort (in person weeks)	Hourly rate	Effort with assets (in person weeks)	Hourly rate
Target tracker	12	21	37	2.4	30
Target tracker, EW devices	6	21	37	5.1	30
Servo controller	16	17	37	1.5	30

Figure 4-4: Recurring Effort in Rosy Market

1. Major changes represent the upper limits of changes for which using the asset is technically more feasible than re-implementing the same functionality using the current process. In our example, these changes are anticipated, but typically require some rewriting of code.

1. See Equation 1 in Chapter 2.

	Number of products, *V*	Current effort (in person weeks)	Hourly rate	Effort with assets (in person weeks)	Hourly rate
Target tracker	4	21	37	2.4	30
Target tracker, EW devices	2	21	37	5.1	30
Servo controller	3	17	37	1.5	30

Figure 4-5: Recurring Effort in Dismal Market

The economies of scope for the good market scenario rounded to the nearest thousand dollars equal $862,000. For the bad market scenario, the economies of scope equal $233,000.

Investment Decision

Once the cash inflows and outflows are established, you are ready to determine the investment value of the asset portfolio for two market scenarios. Substituting the cash flow estimates and dates discussed above for the terms in Figure 4-2 produces the decision tree shown in Figure 4-6. Domain engineering expenses, that is, expenses to build the architecture and assets, were calculated using an average hourly rate of $38. All numbers have been rounded to the nearest thousand.

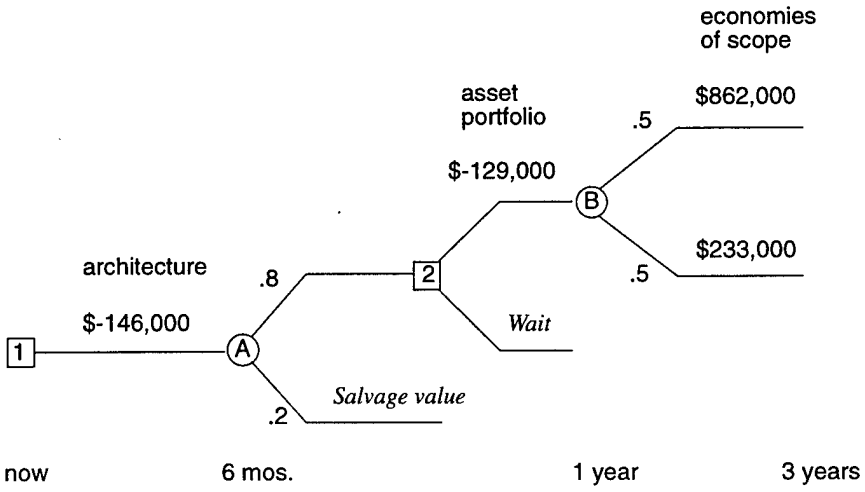


Figure 4-6: Decision Tree with Numbers

To determine whether to invest today, you start at the right side of the tree and work backwards to step 1 on the left. This means first determining what you would do at step 2. The net expected value at step 2 is shown in equation 5. The opportunity cost of capital for the company is 18% (r=.18), and the duration (n) for economies of scope is two years (years 2 and 3).

(5) $$NPV_{step2} = -I_{portfolio} + q(S_{high}e^{-rn}) + (1-q)(S_{low}e^{-rn})$$

Substituting these values in Equation 5 gives the result shown in Equation 6.

(6) $$NPV_{step2} = -129000 + 0.5(862000 \cdot e^{-0.18(3-1)}) + 0.5(233000 \cdot e^{-0.18(3-1)})$$

$$NPV_{step2} = 252578$$

Because the NPV of the upper branch following step 2 is large, the decision to wait is not followed. Deferring an investment is more appropriate when uncertainty is great and the immediate NPV is small. In this case, waiting one year would probably incur an opportunity cost close to half of the NPV of $252,578. The components would not be available for the products built in year 2.

The expected net present value at step 1 is shown in Equation 7. The NPV at step 2 represents the present value of expected benefits for the upper branch at milestone A. Since point A is 6 months from today, the number is discounted at the risk-free rate of 7%.[1]

(7) $$NPV_{step1} = -146000 + 0.8\left(252578 \cdot e^{-0.07(\frac{1}{2}-0)}\right) + 0.2(0)$$

$$NPV_{step1} = 49000$$

Thus the architecture is worth $49,000 solely from the economies of scope of two components.[2] Investing today gives you the option to reduce costs by a weighted average of $252,000. And adding another component would probably increase the benefits significantly.

1. Recall that the technical risk of developing the architecture is modeled in the tree, and the economic uncertainty related to the economies of scope has already been included the NPV at step 2.

2. Note that this value does not include any net benefits (or costs) from using the architecture in other software development and maintenance activities. Cash flows from sources other than component use need to be included, such as the reduction in the time to isolate changes [Stark 93]; in this example they are not. Further research is needed to model the full value of a software architecture.

   Also, this approach has limitations. It values an asset primarily for its contribution to economies in production: how architectural properties, such as information security or system performance, affect revenues or market share are not measured by economies of scope.

The formula and the asset product matrix facilitates sensitivity analysis. In sensitivity analysis, you test how much the NPV will change if one variable is changed. For example, if you relaxed the constraints of an asset so it applied to a greater variety of products, how would the economies of scope change? If the bad market scenario was not as pessimistic, how much would it change the investment decision? What if the hourly expenses for application engineers were greater?

This model assumes that the risk associated with economies of scope is the same as the today's risk of return on the company's products. But, because of advantages in time-to-market, perhaps the risk would decrease: The uncertainty may not be entirely independent of asset portfolio. Also, the spread in the market forecast may be greater or less and it may vary during the two year period. Effort could be spent breaking the products out by time period and modeling the change in risk over time.

Chapter 5 Summary

5.1 Product Strategy

A product strategy describes how a company plans to evolve its product lines over time. It typically describes the customer segments targeted by the company, the products that may be built, including their key differentiating attributes, and the current and expected revenue and market share for each product.

Rather than plan at the tactical level, a product strategy expands the product horizon to multiple years. Instead of thinking about the product to be delivered in 18 months while mired in the details of trying to get the current product out the door, managers consider the variety of products that may improve future market position. Treating those products as if they were to be developed jointly - all at once using assets - managers determine the economies of scope.

This kind of planning compels engineers to identify where their designs must be flexible. It better prepares the organization for shifts in the market. A capability to build products simultaneously is a competitive advantage. Extending the design space from one product to a variety of products increases the chances for rapid delivery and allows for technology growth.

5.2 Approach

The investment analysis approach described in this report identifies assets that maximize economies of scope for a product line. It follows these key steps:

1. Construct an asset portfolio:

    a. Establish the variety of products planned for a product line to establish scope and key product differences. This information usually comes from marketing.

    b. Screen patterns identified in the composition or development of products scoped in step 1. These patterns are identified by engineering, usually by modeling the software system. The patterns are evaluated against criteria that affect economies of scope.

c. Choose assets that fit the patterns screened in the previous step. Current cost drivers, skills, and desired flexibility are considered in the selection.

d. Evaluate portfolio of assets to optimize properties such as budget, scope and impact on current process. The investment team reasons about the results and makes adjustments.

2. Model the investment. This includes defining a deployment strategy to manage uncertainties, and estimating cash inflows and outflows for the portfolio.

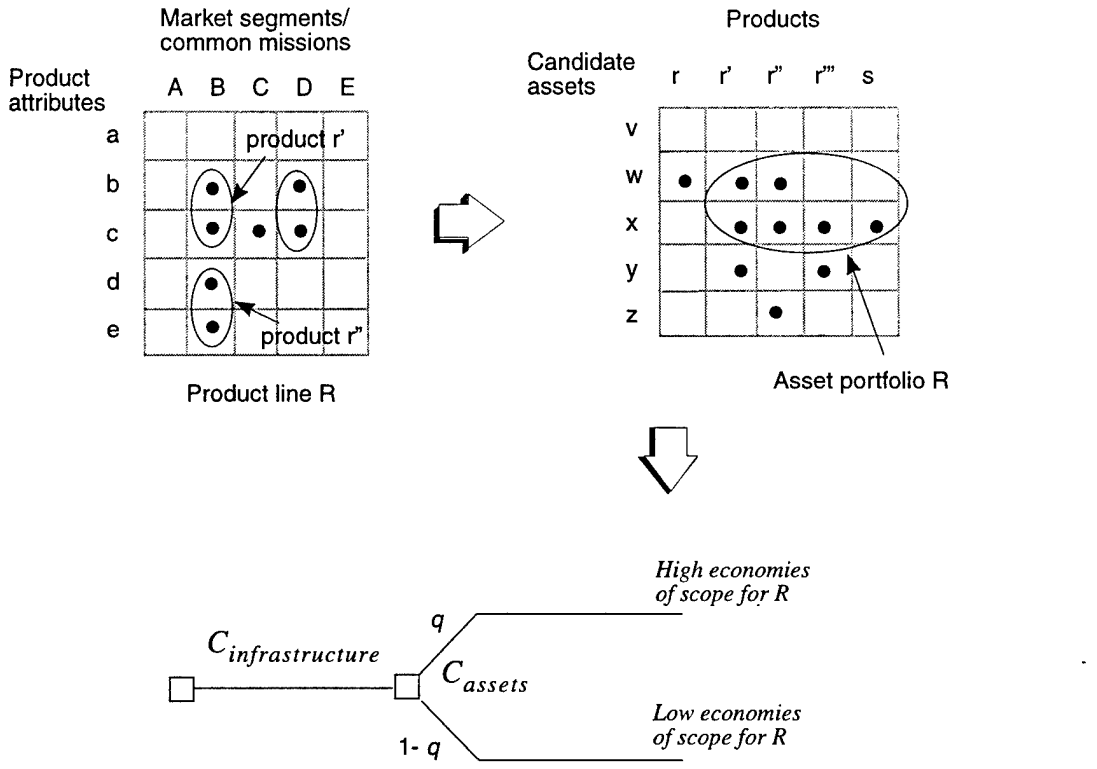


Figure 5-1: Summary of Approach

Figure 5-1 shows a sequence of graphical representations used in the approach. By understanding the product attributes that differentiate products in product line R, assets are proposed that economically accommodate changes to these attributes. A subset that optimizes the non-financial factors affecting economies of scope is included in a portfolio. Economies of scope are then calculated for the portfolio. A decision tree is used to determine the investment value of the portfolio.

5.3 Optimization

The goal of the approach is to define a product strategy such that assets and an application engineering process can be economically developed and used for the variety of products that will be marketed. A firm must find the combination of working capital and inputs (assets) that minimizes the cost and time to produce a maximum variety of products that are closely targeted to different customer segments.

To maximize market share, line managers want to plan products that will personally satisfy each customer. They segment the market and define a variety of products for which there is high demand at different prices. Product lines are the collections of these products.

To reduce costs and time-to-market, technology managers want to build an asset portfolio that can be used to develop many different products in a product family. A product family is defined as the variety of products that can be produced using the assets. The scope of the family depends on the robustness of the abstraction that unifies the assets into a functioning system: a design or architecture, the physics or business rules, a coordination strategy, or the system platform.

The degree to which the cost savings of a product family can apply to the variety of products in a product line can be measured by economies of scope (Figure 5-2). To improve economies of scope, technology managers focus on flexibility (software changes per day) and the tradeoff between skills and automation [Schonberg 86]. They maximize product variety while minimizing the number and variation of assets.[1] Maximum economy occurs when a product family fully supports one or many product lines.

Economies of scope can also inform outsourcing decisions. Software assets are outsourced when the company cannot add as much value as a supplier. This usually occurs when the scope of the asset is much smaller in the company than in the marketplace. A supplier enjoys greater economies because its market is larger, and a greater variety of products gives the supplier more opportunities to nurture and leverage its expertise.

Many companies organize and manage around product lines to ensure that they are focused on maximizing revenue streams and market share. These lines tend to isolate technology: There often is no organizational entity whose control spans the technical commonalities across product lines. Localizing all development in individual product lines impairs synergy and can cause redundancies and inefficiencies across the organization. Commonality of solutions and core technology are neither exploited nor given the critical mass of resources [Prahalad 90].

1. To improve economies of scale, managers typically focus on productivity (lines of code per day) and the tradeoff between labor and capital. They maximize quantity while minimizing variation in production.

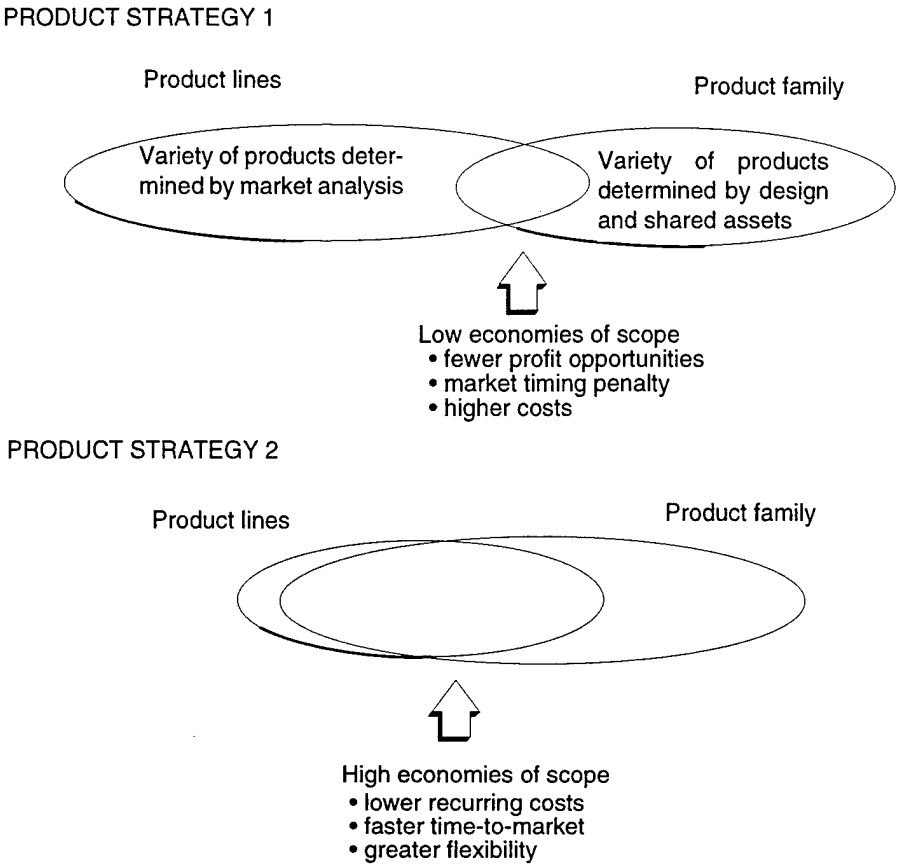


Figure 5-2: Different Product Strategies

Thus, focusing on a product family is required. If technology managers invest in improving the development capability across product lines — improving core technology, extending the asset base, introducing new generations — then previously unavailable opportunities to develop markets become possible. The capability to integrate technologies embodied in software assets in new ways drives marketing to investigate opportunities based on what the company can quickly produce.

The product strategy of the company is no longer solely market driven. The company orients its product lines to market segments that can be supported by a product family. It nurtures the evolution of product generations to anticipate and open new markets. The company can then create shifts in the market and is more ready to meet market shifts created by their competitors. The company will achieve greater economies of scope.

Appendix A References

[Boehm 95] Boehm, B.; Clark, C.; Horowitz, E.; & Westland, C. "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0." *Annals of Software Engineering Special Volume on Software Process and Product Measurement.* Amsterdam, The Netherlands: Baltzer AG, Science Publishers, 1995.

[Boone 89] Boone, L. & Kurtz, K. *Contemporary Marketing,* 6th Edition. Chicago, IL: Dryden Press, 1989.

[Brealey 91] Brealey, R. & Myers, S. *Principles of Corporate Finance.* NewYork, NY: McGraw-Hill, 1991.

[Brownsword 96] Brownsword, L. & Clements, P. *A Case Study in Successful Product Line Development* (CMU/SEI-96-TR-16). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

[Clements 96] Clements, P. & Northrop, L. *Software Architecture: An Executive Overview* (CMU/SEI-96-TR-003). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

[Cusumano 91] Cusumano, M. *Japan's Software Factories.* New York, NY: Oxford University Press, 1991.

[Deming 93] Deming, E. *The Essential Deming Videotapes.* Cambridge MA: Center for Advanced Engineering Study, Massachusetts Institute of Technology, 1993.

[Dewey 95] Dewey, R. *Streamlining Software Development* (Report No. 833). Los Angeles, CA: SRI International Business Intelligence Program, 1995.

[Dixit 94] Dixit, A. & Pindyck, R. *Investment Under Uncertainty.* Princeton, NJ: Princeton University Press, 1994.

[Dixit 95] Dixit, A. & Pindyck, R. "The Options Approach to Capital Investment." *Harvard Business Review 73,* 3 (May-June 1995): 105-115.

[Dumain 89] Dumaine, B. "How Managers Can Succeed Through Speed." *Fortune 119,* 4 (February 1989): 54.

[Gaffney 92] Gaffney, J. & Cruickshank, R. "A General Economics Model of Software Reuse." *Proceedings of International Conference on Software Engineering.* Melbourne, Australia, May 11-15, 1992. New York, NY: ACM, 1992.

[Gamma 94] Gamma, E.; Helm, R.; Johnson, R.; & Vlissides, J. *Design Patterns, Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley Publishing Company, 1995.

[Garlan 94] Garlan, D.; Allen R.; & Ockerbloom, J. "Architectural Mismatch or Why It's Hard to Build Systems out of Existing Parts." *Proceedings of the 17th International Conference on Software Engineering.* Seattle, WA, April 23-30 , 1995. New York, NY: ACM, 1995.

[Guimarães 96] Guimarães, M. *Real Options Tutorial* [online]. Available WWW <URL:http://www.puc-rio.br/marco.ind/tutorial.html> (1996).

[Henderson-Sellers 93] Henderson-Sellers, B. "The Economics of Reusing Library Classes." *Journal of Object-Oriented Programming 6,* 4 (July-August 1993): 43-50.

[HP 93] Hewlett Packard Laboratories. *Company Presentation on Systematic Software Reuse.* Palo Alto, CA: Hewlett Packard Laboratories, 1993.

[Hull 94] Hull, J. *Options, Futures and Other Derivative Securities,* 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1994.

[Huston 88] Huston, J. & Butler, R. "The Location of Airline Hubs." *Southern Economic Journal, 57,* 4 (April 1991): 975-981.

[Kotler 91] Kotler, P. & Armstrong, G. *Principles of Marketing,* 5th edition. Englewood Cliffs, NJ: Prentice Hall, 1991.

[Lanergan 84] Lanergan, R. & Grasso, C. "Reusability in Programming: A Survey of the State of the Art." *IEEE Transactions on Software Engineering SE-10,* 9 (September 1984): 488-494.

[Lim 92] Lim, W. "Cost Justification Model for Software Reuse." *Proceedings of the WISR '92 5th Annual Workshop on Software Reuse.* Palo Alto, CA, October 26-29, 1992. Palo Alto, CA: Hewlett-Packard, 1992.

[Macala 96] Macala, R.; Stuckey, L.; & Gross, D. "Managing Domain-Specific, Product-Line Development." *IEEE Software 13,* 3 (May 1996): 57-67.

[McDonald 95] McDonald, M. & Dunbar, I. *Market Segmentation.* London, England: Macmillian Press Ltd, 1995.

[Meyer 93] Meyer, M. & Utterback, J. "The Product Family and the Dynamics of Core Capability." *Sloan Management Review* (Spring 1993).

[Mili 95] Mili, H.; Mili, F.; & Mili A. "Reusing Software: Issues and Research Directions." *IEEE Transactions on Software Engineering 21*, 6 (June 1995): 528-562.

[Myers 84] Myers, S. "Finance Theory and Financial Strategy ." *Interfaces 14*, 1 (January-February 1984): 126 -137.

[Panzar 81] Panzar, J. & Willig, R. "Economies of Scope." *American Economic Review 71* (May 1981): 268-272.

[Parker 88] Parker, M. & Benson, R. *Information Economics: Linking Business Performance to Information Technology.* Englewood Cliffs, N.J: Prentice Hall, 1988.

[Pindyck 91] Pindyck, R. & Rubinfeld, D. *Microeconomics.* New York, NY: Macmillan Publishing Company, 1991.

[Poulin 93] Poulin, J; Caruso, J; & Handcock, D. "The Business Case for Software Reuse." IBM Systems Journal 32, 4 (1993): 567-594.

[Prahalad 90] Prahalad, C.K. & Hamel, G. "The Core Competence of the Corporation." *Havard Business Review 68*, 3 (May-June 1990): 79-91.

[Rix 92] Rix, M. "Case Study of a Successful Firmware Reuse Program." *Proceedings of the WISR '92 5th Annual Workshop on Software Reuse.* Palo Alto, CA, October 26-29, 1992. Palo Alto, CA: Hewlett-Packard, 1992.

[Roussel 91] Roussel, P.; Saad, K.; & Erickson, T. *Third Generation R&D, Managing the Link to Corporate Strategy.* Boston, MA: Harvard Business School Press, 1991.

[Sanderson 91] Sanderson, S.W. "Cost Models for Evaluating Virtual Design Strategies in Multicycle Product Families." *Journal of Engineering and Technology Managment, 8* (1991): 330-358.

[Schach 94] Schach, S. "The Economic Impact of Software Reuse on Maintenance." *Software Maintenance: Research and Practice, 6* (1994).

[Schonberger 86] Schonberger, J. *World Class Manufacturing*. New York, NY: Macmillan Publishing Company, 1986.

[Stark 93] Stark, Mike. "Impacts of Object-Oriented Technologies: Seven Years of SEL Studies." *Proceedings of OOPSLA '93*. New York, NY: ACM, 1993.

[Ward 95] Ward, A.; Liker, J.; & Sobek II, D. "The Second Toyota Paradox: How Delaying Decisions Can Make Better Cars Faster." *Sloan Management Review 36*, 5 (Spring 1995): 43.

Appendix B Glossary

application engineering
an engineering process that develops a family of software products from partial solutions or knowledge embodied in software assets

deployment strategy
a plan of action for phasing in a portfolio of assets

The process of developing and deploying a portfolio is divided into a sequence of steps that mitigate undesirable events (uncertainties) and capitalize on desirable ones. The sequence of steps constitutes a deployment strategy. The strategy defines the management decision points in a decision tree.

economies of scale
the condition where fewer inputs such as effort and time are needed to produce greater quantities of a single output

economies of scope
the condition where fewer inputs such as effort and time are needed to produce a greater variety of outputs

Greater business value is achieved by jointly producing different outputs. Producing each output independently fails to leverage commonalities that affect costs. Economies of scope occur when it is less costly to combine two or more products in one *production system* than to produce them separately.

investment analysis
a process of estimating the value of an investment proposal to an organization

Investment analysis involves quantifying the costs and benefits of the investment, analyzing the uncertainties, and constructing a spending strategy. This analysis links the strategic and technical merits of an investment to its financial results.

net present value (NPV)
the net value in today's dollars of an investment

NPV is typically calculated by discounting at some interest rate the net income (or loss) that occurs within a time period from an investment.

opportunity analysis

an analysis that compares the costs and benefits of using a resource in one way against the costs and benefits of using the same resource in an alternative way

An opportunity cost is the value of a resource in a competing use. Opportunity analysis discounts the benefit of an approach against its opportunity costs - the benefits of using a resource in an alternative way.

pattern

a partial solution that is replicated from product to product, or knowledge that is reapplied in building each product

The American Heritage Dictionary defines a pattern as "a consistent, characteristic form, style, or method." This is the spirit in which the term is used in this report. Our use of the word expands the concept, yet is consistent with that described in some object-oriented literature. Gamma concentrates on patterns that solve design problems [Gamma 94], while we concentrate on patterns that exist in the composition of multiple products or in the production of multiple products. The objective is to find similarities that reduce engineering tasks and decisions.

portfolio

a group of investments exhibiting desired risk, reward, and other business attributes

A portfolio helps management visualize assets as pieces of an investment strategy. Mangers construct a portfolio by choosing the assets that optimize investment objectives and diversify risks.

product family

the set of different products that can be produced from a common design, shared assets, and an application engineering process

Membership in the set depends on the abstraction unifying the assets into a functioning system: an architecture, the physics or business rules, or the hardware platform. This definition is an elaboration of definition used by Marc Meyer: "[the set of] products that share a common platform [design and standards] but have specific features and functionality required by different sets of customers" [Meyer 93].

product line

1. a group of products that provide a core benefit yet differ along attributes which affect the buying behavior of different customer groups

2. a group of products that are closely related, either because they function in a similar manner, are sold to the same customer groups, are marketed through the same types of outlets or fall within given price ranges

An example of a product line is the variety of camcorders sold by Panasonic. Products are not related in a product line because they share a common implementation; they are related by market and customer criteria. Product lines are lengthened or pruned according to changes in markets, competition and customer preferences. The source of the 2nd definition is *Marketing Definitions: A Glossary of Marketing Terms*, Chicago: American Marketing Association, 1960.

product strategy

an exposition of the products that an organization plans to sell over time

The exposition includes a description of the products and a rationale for why they were selected. The marketing strategy, delivery window, and revenue projections for each product are also discussed.

product variety

1. a set of products that require software changes

2. a variable denoting a set of products considered in investment analysis

production system

a system of people, functions, and assets organized to produce, distribute, and improve a family of products. Two functions included in the system are domain engineering and application engineering.

robustness

the quality of an asset to be changed or extended without other properties either being lost or becoming unpredictable

scope

1. the area or content that something circumscribes

2. a. the breadth of function b. the opportunities of use c. the full extent of variation: range

3. the classes of phenomena included in a software description or a domain

The first definition is the first perspective we take in the approach: We define the variety of products to circumscribe our investigation of patterns. Our viewpoint is a product line, looking in to identify robust commonalities. The second definition is the perspective adopted to analyze the economies of an asset. Our viewpoint is a software asset, looking out at the range of uses. The third definition deals with the ontological content of an asset — what it is and is not. The scope of a world rainfall map is different from the scope of a world population map [Jackson 95].

software asset

a description of a partial solution (such as a component or design document) or knowledge (such as a requirement database or test procedures) that application engineers use to build or update software products

Software assets codify *patterns* in a tangible form so that engineers can use them. Typically assets are components and tools that engineers use to proliferate products. Examples include architectures, code generators, design templates, product technology notebooks, core subsystems, implementation standards, and test suites.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS
None

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for Public Release
Distribution Unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
CMU/SEI-96-TR-010

5. MONITORING ORGANIZATION REPORT NUMBER(S)
ESC-96-TR-010

6a. NAME OF PERFORMING ORGANIZATION
Software Engineering Institute

6b. OFFICE SYMBOL (if applicable)
SEI

7a. NAME OF MONITORING ORGANIZATION
SEI Joint Program Office

6c. ADDRESS (city, state, and zip code)
Carnegie Mellon University
Pittsburgh PA 15213

7b. ADDRESS (city, state, and zip code)
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

8a. NAME OFFUNDING/SPONSORING ORGANIZATION
SEI Joint Program Office

8b. OFFICE SYMBOL (if applicable)
ESC/AXS

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F19628-95-C-0003

8c. ADDRESS (city, state, and zip code))
Carnegie Mellon University
Pittsburgh PA 15213

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT NO.
63756E	N/A	N/A	N/A

11. TITLE (Include Security Classification)
Investment Analysis of Software Assets for Product Lines

12. PERSONAL AUTHOR(S)
James Withey

13a. TYPE OF REPORT
Final

13b. TIME COVERED
FROM TO

14. DATE OF REPORT (year, month, day)
November 1996

15. PAGE COUNT
60

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (continue on reverse of necessary and identify by block number)
software reuse (systematic), return on investment, product line development, economies of scope, software assets, production system, capital budgeting, business case analysis, portfolio analysis, net present value, dynamic discounted cash flow analysis, investment uncertainty and risk, product families,product lines, technology transition, decision trees

19. ABSTRACT (continue on reverse if necessary and identify by block number)

Group, product line, and program managers are faced with allocating resources to projects. Should all resources be dedicated to meet near-term deliverables? Or should some be siphoned off to build software assets that may improve quality, flexibility, and reduce cost and time-to-market of future products in the product line? These managers also have to determine which assets to buy or build. The choices are many, ranging from reusable code components to design models to application generators, and each has a different risk and cash flow profile.

This report introduces an approach that will help managers make these allocation decisions. The

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ■ SAME AS RPT ☐ DTIC USERS ■

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified, Unlimited Distribution

22a. NAME OF RESPONSIBLE INDIVIDUAL
Thomas R. Miller, Lt Col, USAF

22b. TELEPHONE NUMBER (include area code)
(412) 268-7631

22c. OFFICE SYMBOL
ESC/AXS (SEI)

ABSTRACT — continued from page one, block 19

report outlines a planning and communication tool for analyzing investments in software assets for product lines.

Although the report is not a guidebook, the concepts, criteria, and investment modeling techniques will be useful in making and justifying proposals for funding. The concepts are drawn from the fields of microeconomics, corporate finance, marketing, R&D technology management, and software reuse.